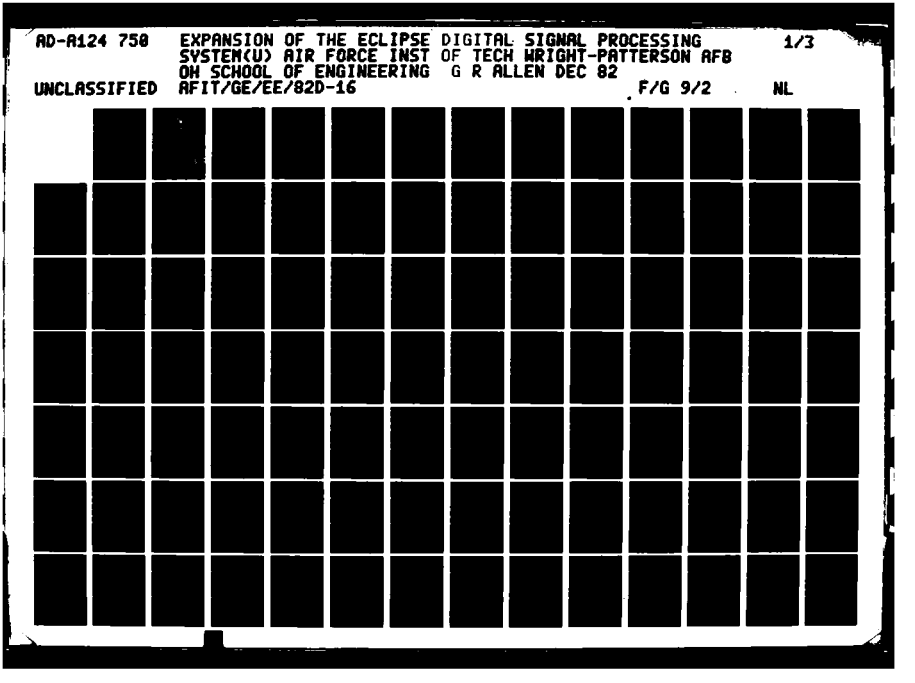
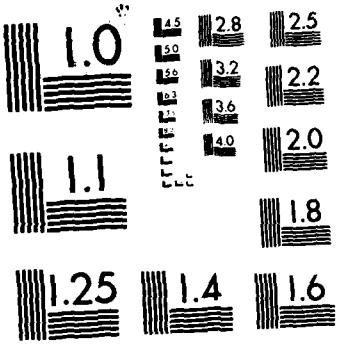

AD-A124 750 EXPANSION OF THE ECLIPSE DIGITAL SIGNAL PROCESSING 1/3
SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB
OH SCHOOL OF ENGINEERING G R ALLEN DEC 82
UNCLASSIFIED AFIT/GE/EE/82D-16 F/G 9/2 NL



EXPANSION OF THE ECLIPSE DIGITAL SIGNAL PROCESSING SYSTEM

THESIS

AFIT/GE/EE/82D-16 Gordon R. Allen
1st Lt USAF

UNITED STATES AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY
Wright-Patterson Air Force Base, Ohio

EXPANSION OF THE ECLIPSE DIGITAL SIGNAL PROCESSING SYSTEM

THESIS

AFIT/GE/EE/82D-16 Gordon R. Allen
1st Lt USAF

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFIT/GE/EE/82D-16	2. GOVT ACCESSION NO.: AD-A124750
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE *(and Subtitle)*: EXPANSION OF THE ECLIPSE DIGITAL SIGNAL PROCESSING SYSTEM	5. TYPE OF REPORT & PERIOD COVERED: MS Thesis
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Gordon R. Allen	8. CONTRACT OR GRANT NUMBER(s):
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Air Force Institute of Technology(AFIT-EN) Wright-Patterson AFB, Ohio 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Institute of Technology(AFIT-EN) Wright-Patterson AFB, Ohio 45433	12. REPORT DATE: December 1982
	13. NUMBER OF PAGES: 247
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*:	15. SECURITY CLASS. *(of this report)*: Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT *(of this Report)*
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES
Approved for public release; IAW AFR 190-17

Approved for public release: IAW AFR 190-17.
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

4 JAN 83

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*
Eclipse Minicomputer
Fast Fourier Transform (FFT)
Digitizing Operations
Filter Design

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*
A signal processing software package was generated for a Data General Eclipse S/250 minicomputer. The model 4331 A/D/A converter was utilized to perform general purpose A/D/A operations and to collect, edit, and play back speech data files. The model 130 array processor was used to perform high-speed convolution and Fourier Transform related operations. The Parks-McClellan algorithm was implemented to allow design of

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

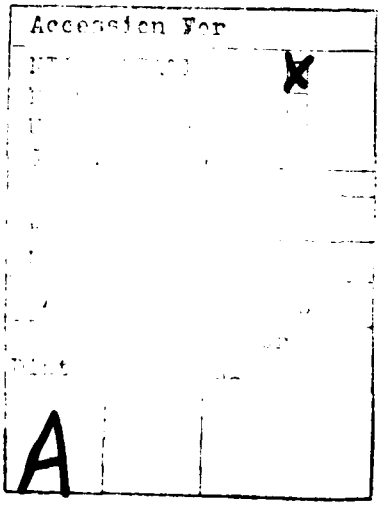
linear phase, finite impulse response filters. Self-explanatory interactive programs for data collection and filter design, together with single line commands for signal processing functions, make this a simple to operate, versatile package for digital signal processing.

AFIT/GE/EE/82D-16

EXPANSION OF THE ECLIPSE DIGITAL SIGNAL PROCESSING SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science



by
Gordon R. Allen, B.S.
1st Lt USAF
Graduate Electrical Engineering
December 1982

Preface

The data base at the Air Force Institute of Technology (AFIT) signal processing laboratory has not been able to keep pace with recent hardware expansion. An array processor has been installed that has only seen limited application. Its computational speed could greatly increase the speed of many algorithms used in the laboratory. The digitizer that has been added can be controlled by software to a larger degree and can operate on larger data files than the current model. However, it has never been interfaced with other laboratory equipment. The time required to become familiar with these devices is prohibitive to thesis students and other personnel who would benefit most from their use.

This effort resulted from a suggestion by Major Larry Kizer, Assistant Professor of Electrical Engineering at AFIT. Major Kizer teaches the school's digital signal processing course and is primarily responsible for the laboratory's growth. He felt it would be useful to have a software package that integrated these hardware additions into the system. Also, this should be done in such a way as to allow easy operation. With this initial objective, the final result was a software package capable of performing sophisticated signal processing functions, yet very simple to operate.

Gordon R. Allen

Contents

Contents

List of Figures

List of Figures

List of Tables

Abstract

A signal processing software package was generated for a Data General Eclipse S/250 minicomputer. The model 4331 A/D/A converter was utilized to perform general purpose A/D/A operations and to collect, edit, and play back speech data files. The model 130 array processor was used to perform high-speed convolution and Fourier Transform related operations. The Parks-McClellan algorithm was implemented to allow design of linear phase, finite impulse response filters. Self-explanatory interactive programs for data collection and filter design, together with single line commands for signal processing functions, make this a simple to operate, versatile package for digital signal processing.

EXPANSION OF THE ECLIPSE DIGITAL SIGNAL PROCESSING SYSTEM

I Introduction

Background

With the introduction of the Fast Fourier Transform (FFT) in 1964, digital signal processing took on new significance. The Fast Fourier Transform provided an efficient method of calculating the Discrete Fourier Transform (DFT) and made it a much more feasible tool for use in signal analysis. Recent advancements in digital hardware and computer architecture have made digital signal processing techniques even more practical.

Today, digital signal processing techniques have seen application in many fields of study. Here at the Air Force Institute of Technology (AFIT) signal processing laboratory, they are used to investigate pattern recognition problems in the speech and video areas. The laboratory is used to support research by AFIT personnel and other Air Force organizations.

Recent hardware improvements have been made to the laboratory. An array processor and an additional digitizer have been installed. A group of filter design programs has also been procured. However, software has not been generated to

allow these devices and programs to be operated easily. Research efforts could be done in a more timely manner if this software was available.

Summary of the Current System

The AFIT signal processing laboratory contains two Data General minicomputers, the Eclipse S/250 and the Nova 2/10. The Nova computer is interfaced with a digitizer for collecting data samples from an analog signal (A/D operations) and to output data samples (D/A operations). The Eclipse computer, however, is more computationally powerful and has been equipped with its own digitizer and an array processor. The operating options on the new digitizer are software controllable and proper software would allow this device to perform a variety of digitizing operations. Also, due to the extended memory feature of the Eclipse, the Eclipse A/D/A device has the potential of operating on larger data files than the Nova A/D/A device. The array processor could be used to greatly speed up many algorithms that currently require hours to run on the system. However, the Eclipse A/D/A device has not been operated or even interfaced with other laboratory equipment and the array processor has only seen limited application due to the time required to become familiar with these devices.

There is not any software currently available in the laboratory to allow personnel to perform convolution or DFT operations without generating software. To perform signal

processing functions, the Interactive Laboratory System (ILS) and Data General (DG) software packages are available. These packages provide subroutines that the user can apply in programs to perform signal processing operations. However, learning to use these packages is a time-consuming obstacle and it should not be necessary for individuals who only want to perform basic signal processing operations.

A group of machine-portable programs for digital signal processing has been procured for the laboratory. The source code for these programs cannot be compiled and loaded in their present form. The mainline and subroutines of each program need to have machine-dependent variables defined and be compiled separately before the program will operate on the Eclipse.

Objectives

The overall objective of this effort is to create a user-oriented signal processing software package for the AFIT signal processing laboratory. The package will make use of system features that reduce user inputs and will implement the recent hardware improvements. With only basic knowledge of system operation, personnel will be able to do meaningful signal processing operations.

To enhance the laboratory's A/D/A capability, software will be generated that operates the Eclipse A/D/A device and makes use of the Eclipse's extended memory feature. The various ways to use this device will be studied. Those most useful to signal processing applications will be organized into a user's manual that will explain how to write software to

operate the device. Two interactive programs will be generated. One program will be designed specifically to handle speech data operations, while one will be designed to flexibly operate all of the device software controllable features to handle peculiar digitizing operations.

To allow personnel to directly perform basic signal processing operations with the array processor, general purpose signal processing programs will be generated. These programs will allow the user to operate on entire data files by typing a single line command. They will also serve as examples of how to use the array processor to perform the basic operations that are done in related ILS and DG subroutines.

To set up a computer-aided filter design capability in the laboratory, the Parks-McClellan algorithm for designing linear phase finite impulse response (FIR) filters will be implemented. This is one of the programs contained in the group of machine-portable programs. This program can be used to design a wide range of lowpass, highpass, and multiband filters. It also can be used to design differentiators and Hilbert transformers. In addition to covering a wide range of filter applications, this effort will uncover any problems that might exist in implementing other programs in this group on the Eclipse. Additional software will be generated to allow this program to be more self-explanatory and easily operated.

II A/D/A Operations

This chapter describes the Eclipse A/D/A device. The capabilities of the device and the operating methods of most interest for signal processing applications will be discussed. The two programs will be presented that use this device to operate on speech data and to perform general purpose digitizing operations.

The Eclipse A/D/A Device

The Eclipse computer in the AFIT signal processing laboratory is equipped with a model 4331 analog data subsystem (Ref 1; Ref 2) and the Sensory Access Manager (SAM) software package (Ref 3). The SAM software package is a Data General package that aids in building I/O programs for Data General computers equipped with A/D/A devices. The software package contains libraries that can be used to manipulate the device.

The model 4331 subsystem is a general purpose A/D/A device with a resolution of 12 bits. The A/D section has an A/D converter with two multiplexors that allow 16 channels of differential input. Data samples can be collected from a single channel or a sequential list of channels. The D/A section contains two independent D/A converters. The A/D and D/A sections have both been set to operate at the $\pm$5v range and to handle conversion values in a two's complement format. Each 12-bit conversion value is stored in one 16-bit machine word. The remaining least significant four bits of

a word are not used. A more detailed description of this device and how to write software to operate it, is presented in the Eclipse A/D/A Device User's Manual which is attached as Appendix A.

One problem with the device was not resolved. A single conversion operation, according to the specification, should be able to handle up to 16,384 samples (Ref 3:5-6). However, the device gave an error for any conversion operation above 16,073 samples. After an extensive search through the manuals, a user error could not be found to explain this. The error code returned, 2194, indicated an attempt to move conversion data outside the area set up to hold data. According to the SAM User's Manual, this was an error for assembly language operation only and should not occur for Fortran operation. This error occurred, however, for both operations. It was concluded that this was a problem with the device. An option was included in the general purpose program to be discussed later, that allows the maximum error-free conversion count for any specified conversion operation to be quickly found. Using this option, it was noted that this problem existed regardless of the channel or clock source used. The only clock source not used with this option was the pulse generated clock, which is more difficult to set up and would not typically be used for signal processing applications. This option could be used to verify correct operation when the device is repaired.

Memory Management Techniques

This section describes programming techniques that can be used to operate the Eclipse A/D/A device. An executable program on the Eclipse or Nova in the AFIT signal processing laboratory must be 32KW or less. This includes the source code, overhead code, and variable space (Ref 4:1-4). Since operating on most data files usually requires a large amount of variable space, the method used to implement the device in a program is an important consideration due to memory constraints.

The variable space to hold the conversion values of a single conversion operation must be declared in the main program. This would require 16KW of integer array space to hold the maximum specification number of 16,384 samples for a single conversion operation. The memory problem is compounded if it is desirable to do both, A/D and D/A operations, in the same program. The same data arrays cannot be used to both input and output data, since they must appear in different labeled common blocks for an A/D or D/A operation. The additional SAM library overhead further reduces the space left in the main program. Although 32KW certainly provides enough space to allow a program to handle either an A/D or D/A conversion operation with 16,384 samples, there may not be enough space left for the rest of the user's source code. To remedy such situations, Data General Fortran V provides two methods, overlays (Ref 4:4-1) and swaps (Ref 4:4-4), to increase the source code of the main program. Basically,

program swaps operate by overwriting main memory with a new program, while overlays overwrite only a section of main memory with new code. Using one of these methods, a secondary program can be used by the main program to perform A/D/A operations. Since the secondary program is usually quite large due to large data arrays, a program swap will generally be the best method to use. If this is the case, parameters specifying the A/D/A operation can only be passed to the secondary program by writing them to a disk file. The secondary program must then read these parameters from the disk file. This is the method used by the programs in the next two sections which use separate, secondary programs to handle A/D and D/A operations.

Sampling at 8KHz, the single conversion operation maximum of 16,384 samples would provide only 2.05 sec of speech data. It is desirable to work with longer speech files. Fortunately, the Eclipse computer has a feature called extended memory mapping (Ref 4:4-11), which allows large amounts of data to be moved quickly.

Extended memory mapping can be visualized as follows. A window is set up in the main program that can be made to slide along additional memory called extended memory. Data can be routed to and from extended memory through this window. Actually, data is not physically moved, address registers are simply changed. The Eclipse computer has up to 42KW of additional memory that can be accessed through remapping.

The extended memory setup that is used in the speech

program of the next section is shown in Fig 2-1. In this setup, conversion operations are performed in 10,240-sample sections. The results of the first four conversion operations are routed through the window into extended memory. Using the data buffer to hold the fifth conversion operation results, up to 51,200 samples can be collected. Sampling at 8KHz, this provides 6.40 sec of speech data.

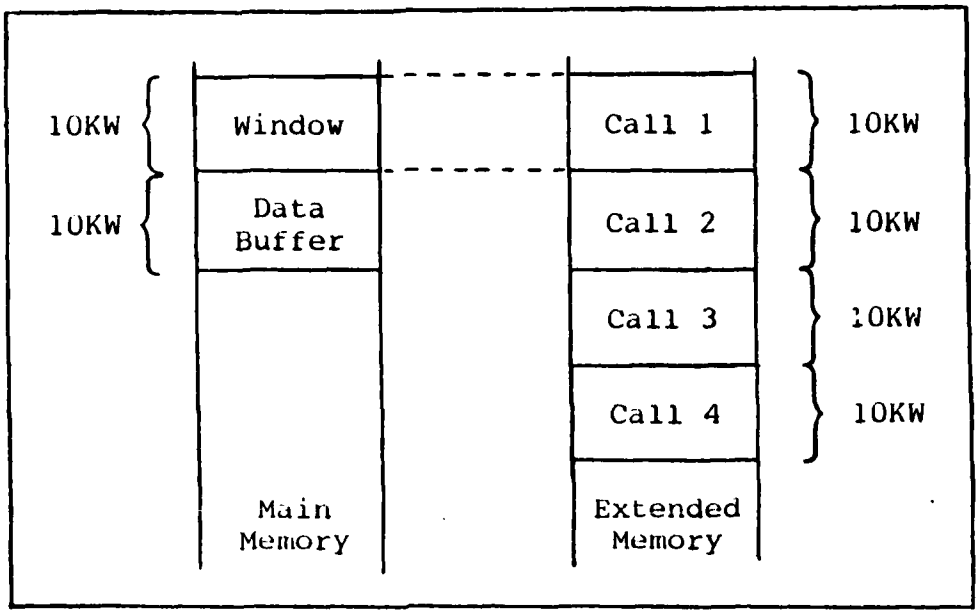


Fig 2-1 An extended memory setup for repeated conversion operations

Of course, each of the remap operations causes a delay where sampling points may be lost. A test was devised to give an approximate indication of the number of points that may be lost in the above setup. For the test, a triangle wave was sampled using the setup shown in Fig 2-1. The break in the linearity of the wave during the remap operation was noted to determine the number of points lost. The Eclipse

computer has two user terminals, referred to as foreground and background, that share the computer's single CPU. To note the affect of CPU activity on the time required to perform a remap operation, the test was conducted on the background for three conditions. First, the foreground was made inactive*, allowing all of the CPU's attention to be given to the A/D/A program. For the other two test runs, the foreground was allowed to be active. On the second test run the foreground set idle and on the third test run it was used to compile a program. The test setup and description is given as Appendix B. The affect of the remap operation on the sampled triangle wave is shown in Figs 2-2, 2-3, and 2-4. For these Figs, a

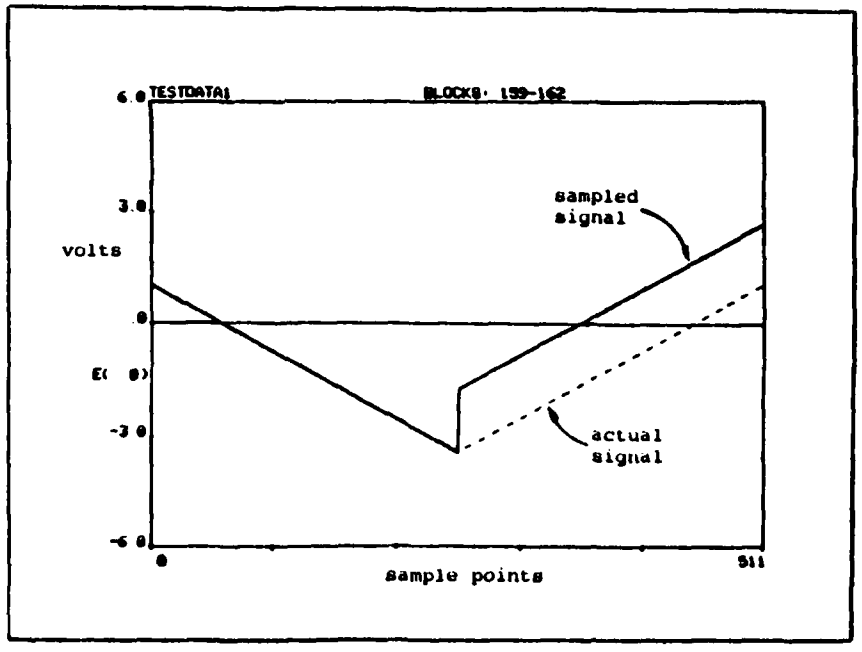


Fig 2-2 Extended memory data collection results while foreground was inactive

* Inactive implies that the foreground was shut down with the CTRL-F command.

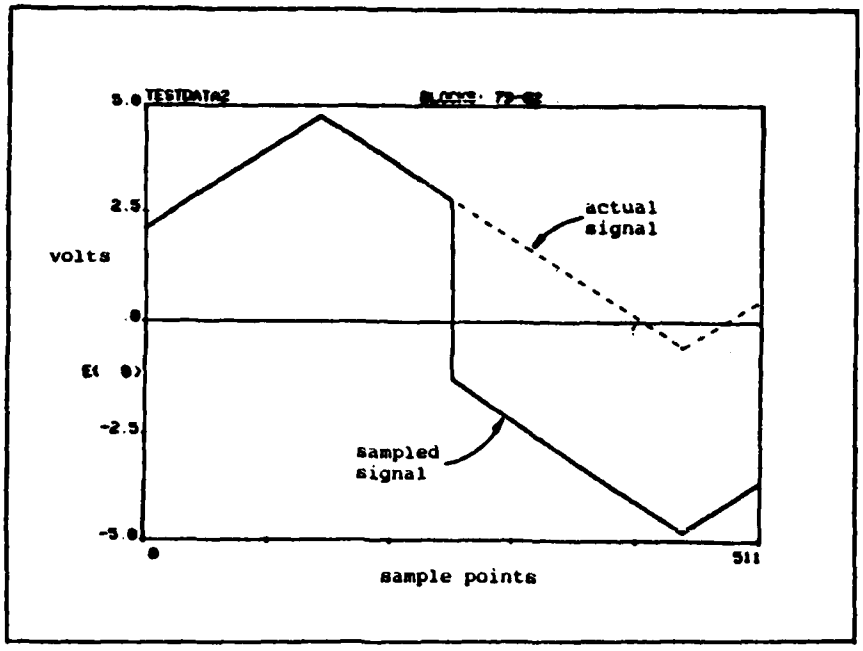


Fig 2-3 Extended memory data collection results while foreground was idle

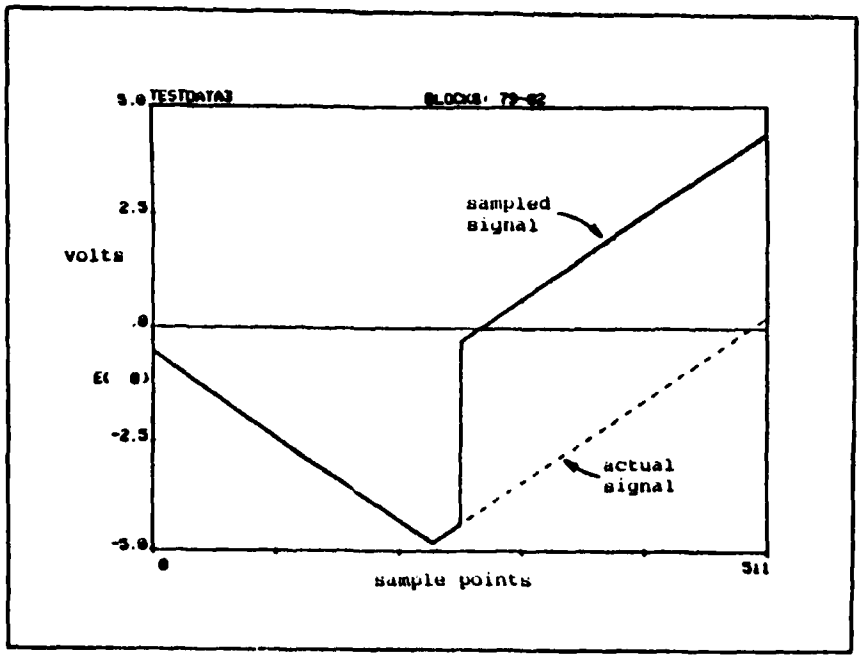


Fig 2-4 Extended memory data collection results while foreground was compiling

set of sample points were chosen that conveniently illustrated the break in linearity.

The number of points lost during the remap operation in each of the plots was calculated as follows. The change in voltage between all data points, except where the slope changes and the remap transition occurs, was computed and averaged. The following formula was used to compute the points lost,

$$Pts = \frac{Mag}{Incr} - 1$$

where Pts = points lost
Mag = voltage magnitude of remap transition
Incr = average voltage change between sample points

A program was used to do the above calculations and the results for the three plots are shown in Table 2.1.

Filename	DATA1	DATA2	DATA3
Disk Blocks	80-81	40-41	40-41
Pts	96	221	224
Mag	1.6895	4.0601	4.1162
Incr	.0174	.0183	.0183

Table 2.1 Remap operation test results

The results indicate that not the level of activity on the opposite terminal, but whether it is active or inactive, is the prime determinant of the points that will be

lost. The test data was collected at a 21KHz sampling rate, however, speech is usually sampled at only 8KHz. Since the sampling rate and the number of points lost is known for each condition in the test runs, the time required for the remap operation and the number of points lost for sampling at 8KHz can be easily computed. These results are shown in Table 2.2.

	Condition 1	Condition 2	Condition 3
Remap Interval	4.57msec	10.5msec	10.7msec
Points Lost @ 8KHz	37	84	85

Table 2.2 Remap interval and points lost at 8KHz Sampling

Losing the number of points on the order shown in Table 2.2 could add another dimension of uncertainty in pattern recognition programs. If the remap occurred during the crucial utterance of a short word, its signature could be seriously eroded. Care must be taken using this setup to collect data, not to have utterances during the remap interval. The primary use of such a setup should be to play back edited speech files that have been pieced together such that no utterance occurs during the remap interval. However, even if this does occur during playback, the only affect is the annoying silent gap of the remap interval.

The following two sections describe two programs that were generated using this device. The source code for these

programs and a data format conversion program is given in Appendix C. The source code for the user subroutines named in these programs is included in Appendix F.

A Program for Speech Application

An interactive program for working with speech data was generated. This program allows the user to collect, edit, piece together, and play back speech files. The program can be operated in either short mode, to work with 15,872 samples, or in long mode, to work with 51,200 samples. During execution, the program maintains two buffers on disk file, the data buffer and edit buffer. The data buffer is where conversion values must originally be placed, either by an A/D operation or reading from a disk file. The data buffer can then be placed in the edit buffer where sections of data can be played back and deleted. Editing operations performed on the edit buffer do not affect the data buffer.

The program is actually a collection of six programs where the secondary programs are called upon by swapping. The central program, SPEECH, performs operations on the data buffer and calls up the editing mode. A copy of its main menu options is shown in Fig 2-5. The second program, EDITOR, performs editing operations on the edit buffer and can return to the central program. It provides two types of histograms, by voltage and by block, on specified blocks of data. A copy of the display shown for the same blocks of a speech file is given for each histogram in Figs 2-6 and 2-7. Two of the programs, SMALLIN and SMALLOUT, are used by the central pro-

```
Please select which operation will be performed,
   1: A/D conversion into data buffer
   2: D/A conversion out of data buffer
   3: editing
   4: read from file to data buffer
   5: write data buffer to file
   6: copy data buffer to edit buffer
   7: exit
selection:
```

Fig 2-5 Program SPEECH main menu options

```
Voltage Histogram
blocks:  1- 16     samples:  4096.     total clips      0.
max voltage:  1.5796( 10352)
min voltage: -2.2705(-14880)

     Voltage        Positive       Negative        Total
    Magnitude       Samples        Samples        Samples

      5.0-4.5           0.             0.             0.
      4.5-4.0           0.             0.             0.
      4.0-3.5           0.             0.             0.
      3.5-3.5           0.             0.             0.
      3.0-2.5           0.             0.             0.
      2.5-2.0           0.             1.             1.
      2.0-1.5           2.             9.            11.
      1.5-1.0          14.            12.            26.
      1.0- .5          79.            57.           136.
       .5- .0        1211.          2806.          4017.

Please select which operation will be performed,
   1: D/A conversion of histogram blocks
   2: delete histogram blocks from edit buffer
   3: return to the editing menu
selection:
```

Fig 2-6 Program EDITOR voltage histogram display

```
Block Histogram
blocks:  1- 16     samples:  4096.     total clips      0.
max voltage:  1.5796( 10352)
min voltage: -2.2705(-14880)

     Block        Total        Max
     Number       Clips     Magnitude

        1           0.        1.1890
        2           0.         .5884
        3           0.        1.5796
        4           0.        1.4429
      5-  6         0.        1.8896
      7-  8         0.        2.2705
      9- 10         0.         .8398
     11- 12         0.         .1733
     13- 14         0.         .1489
     15- 16         0.         .0732

Please select which operation will be performed,
    1: D/A conversion of histogram blocks
    2: delete histogram blocks from edit buffer
    3: return to the editing menu
selection:
```

Fig 2-7 Program EDITOR block histogram display

gram to handle short mode A/D operations. The last two programs, BIGIN and BIGOUT, are used by the central and editing programs to handle long mode D/A operations. Each of the four A/D/A programs contains an option to repeat the conversion operation while within the program. This allows the conversion operation to be repeated without the annoying wait that is required to swap in an A/D/A program.

The program that operates the Nova A/D/A device for work with speech data, AUDIOHIST, scans the chosen data file blocks for histogram parameters each time a histogram is requested. This causes an annoying delay between histograms. During editing, histograms may be requested several times to isolate a single block of data that will be deleted. To allow the histograms to be presented quickly, parameter arrays were used in program EDITOR. The first time a histogram is requested, the histogram parameters for the entire edit buffer are collected and stored in parameter arrays. As blocks of data are deleted, the parameter arrays are updated. Subsequent histograms are displayed without any noticeable delay.

A Program for General Purpose Application

An interactive program for performing general purpose digitizing operations was generated. Due to the general nature of its design, the program is rather cumbersome to operate. It is intended to be used to test A/D/A system setups and to handle digitizing operations not routinely performed. It could also be useful when time does not permit

generating additional software for a digitizing operation.

As with the speech program, this program is actually a collection of programs where the secondary programs are called upon by swapping. The central program, DIGITIZE, is very short and simply directs the user to choose either A/D or D/A mode. The A/D program, INDIGI, and the D/A program, OUTDIGI, maintain separate 16KW data buffers. In addition to performing conversion operations, each program allows the user to view, print, or write to disk file specified sections of the data buffer. The D/A data buffer can also be filled by reading from a disk file. Since the data buffers are independent, the only way the contents of one can be placed in the other is through an intermediate disk file.

The D/A program allows the data buffer to be demultiplexed. This option can be used to recover data that was collected on a single channel when the channel scan feature was used in the A/D operation. The starting data sample and the number of samples to be skipped between subsequent saved samples are specified by the user. The channel scan feature is useful for collecting data where the time relationship between signals is important. Shown in Fig 2-8 is an example of such an application. Here the user had recorded analog data simultaneously on various channels of a multi-channel tape recorder at a location outside the laboratory. One of the channels contained a timing signal that was modified and used as an external clock source. Using program DIGITIZE, the analog data on two of the channels was first digitized

using the channel scan feature set to alternate between two channels. Then demultiplexing operations were done on the sampled data to create the data file for each signal. Program CNVRT, which is included in Appendix C, was used to convert the data files into real number format. Program PLOT, which is included in Appendix F, was used to obtain the plot shown. The figure shows the plot of an EEG signal of a dog's brain and a signal used to control the frequency of a strobe light flashed into the dog's eyes. The separate data files were necessary to view the signals and permit data processing.

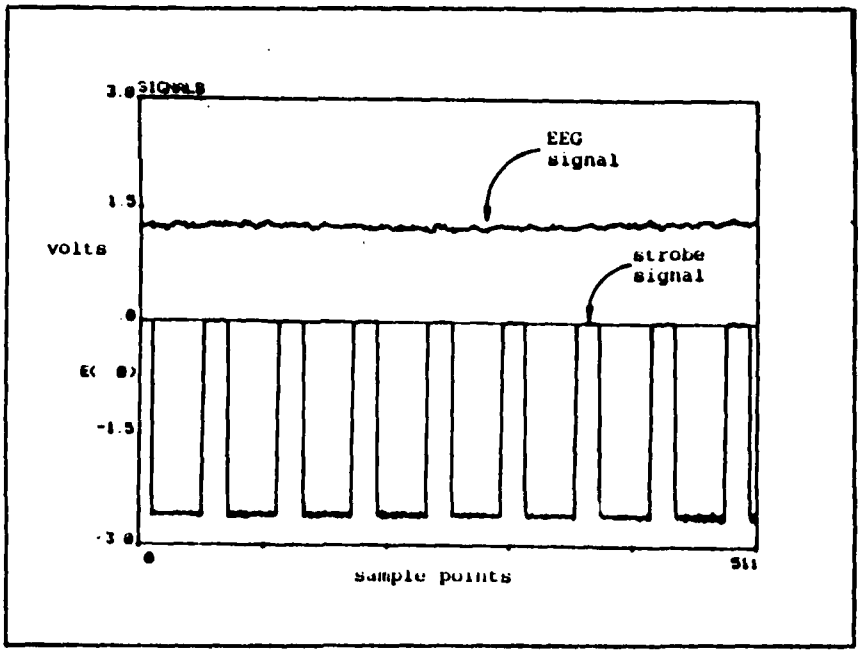


Fig 2-8 An example of recovering multiplexed data with program DIGITIZE

III Signal Processing Functions

This chapter describes the Eclipse array processor. The two methods of utilizing array processor memory will be discussed. Several general purpose signal processing programs that make use of this device will be presented. The source code for these programs and related user subroutines is included in Appendices D and F, respectively. Each program is activated by typing a single line command that identifies the processing function and the data files to be operated on.

The Eclipse Array Processor

The Eclipse computer in the AFIT signal processing laboratory is equipped with a model 130 array processor (Ref 5). The array processor is designed to provide high-speed matrix computations. It contains independent multiply and add/subtract units that can operate simultaneously. Each unit features pipeline design, which allows several operations to overlap one another during the same time period.

There are a variety of matrix operations that can be performed. Each matrix operation is called as a subroutine with a single argument that references a control block. The control block is set up prior to calling the matrix operation by using one or more additional library subroutines. It specifies the operation's parameters, such as the location of input and output data in array processor memory.

Array Processor Memory Management

The array processor contains 4KW of memory which is basically used as a scratchpad for matrix operations. Input data is loaded into this area and output data is retrieved from it. Data can be transferred to and from array processor memory in two ways, directly (Ref 5:2-33) and/or by mapping (Ref 5:2-10).

Moving data directly requires a separate subroutine call to transfer data. It allows data of any length or location in main memory to be moved to any location in array processor memory and vice-versa. The major drawback with this method is the manual loading or unloading of data required for each matrix operation.

With memory mapping, data is transferred automatically. Any contiguous 1KW-multiple of array processor memory can be mapped into main memory. The mapping operation that does this sets up a window in main memory. The data arrays identified in this window can be treated as if they were located in array processor memory. Loading data into these arrays places the data directly into array processor memory. If the array processor window is remapped, then the data in the previous window is destroyed. For this reason, as is the case with extended memory mapping, the mapping operation is usually only called once in the program. Data can be moved to or from the window by setting up a loop that exchanges values with another array outside the window or by performing disk read/write operations on the window data arrays. Of the programs

to be presented, the first performs data transfers directly and the others perform data transfers by mapping.

A Program for Time-Domain Processing

To allow signal processing in the time-domain, a convolution program, CONV, was generated that makes use of the array processor. The program convolves an input file containing up to 32,767 disk blocks with a filter file containing up to 512 points. Since the filter file must remain in array processor memory throughout the operation, a filter file of 512 points would consume one-quarter of array processor memory (4KW = 2048 real points). For this reason, a 512-point maximum was chosen as a balance between memory used to hold the filter and a reasonably large impulse response. The coefficients of many infinite impulse response filters are usually small beyond this length.

The algorithm operates by breaking the input data into sections and using subroutine VCONRZ to convolve each section with the filter response. The overlap-save method is used to piece together the individual matrix operation outputs to form a linear convolution (Ref 6:113-115). The overlap-save method is illustrated in Fig 3-1. The input data is broken up into M-point sections and convolved with an N-point impulse response. The first N-1 points of the first section are zero filled and the first N-1 points of each subsequent section are identical to the last N-1 points of the preceding section. The first N-1 points of each output section are incorrect, while the remaining points are identical to that of a linear

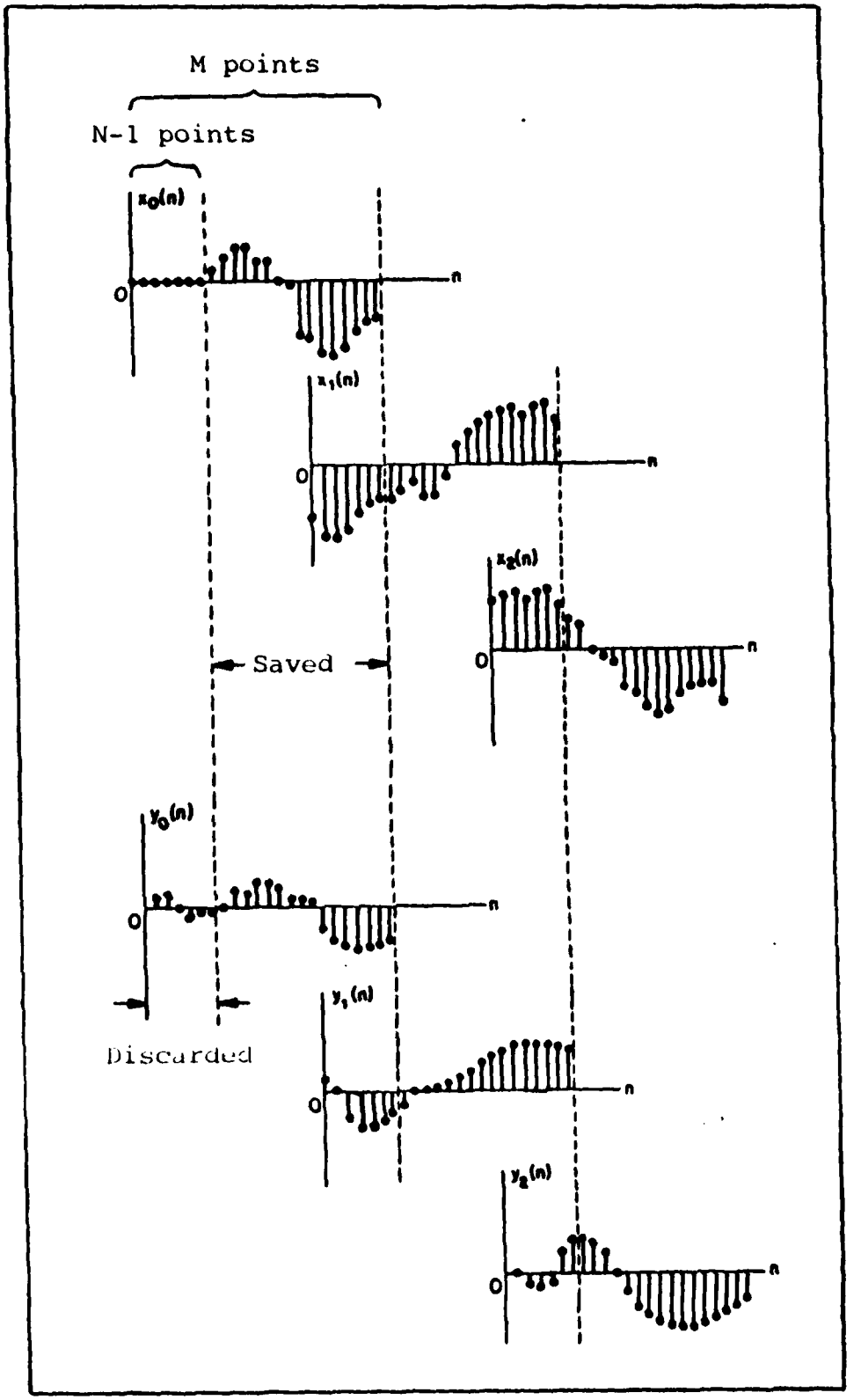


Fig 3-1 The overlap-save method of convolution

convolution. Each output section has its first N-1 points discarded. The remaining sections are then abutted together to construct the final filtered output.

The program makes use of the in-place convolution feature (Ref 5:4-18). Using this feature, the size of each output section is maximized. The data setup for a single operation with an N-point filter file is shown in Fig 3-2a. The filter file must be loaded at the top of array processor memory. The input data must then be loaded N points below the filter data. Using the in-place convolution feature, it can be specified to begin writing output data beneath the filter data and to continue overwriting the input data if necessary. The value of M is determined from the relationship, M = 2048-2N. As shown in Fig 3-2b, for an M-point section of input data, the matrix operation only gives the first M points of the M+N-1 point convolution. Since the program always discards the first N-1 points, each output save section is only M-N+1 points long. If the in-place feature is not used, a convolution operation cannot be specified where output data will overwrite input data.

For a 512-point filter and input file, the program requires two matrix operations to give the linear convolution. First, the front of the input data file is augmented with 511 zeros. The back of the input data file is also augmented with zeros to allow the matrix operation to overrun. The input data is then loaded in two 1024-point sections overlapping each other by 511 points as described in the

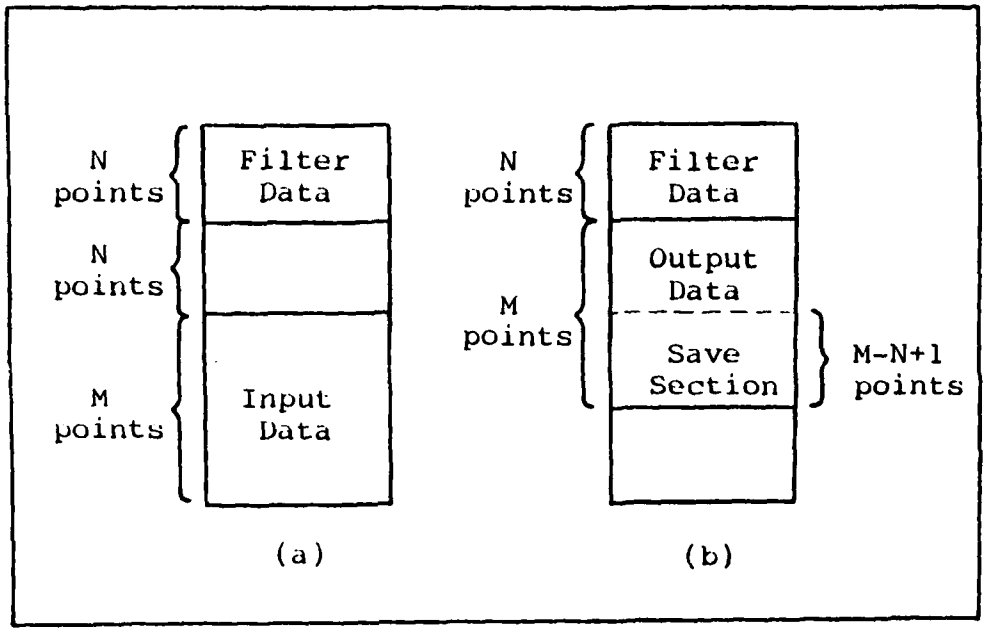


Fig 3-2 Data setup in array processor memory
(a) prior to convolution operation and
(b) after convolution operation

overlap-save discussion. Each convolution operation yields 513 points of the linear convolution. The last three points of the second convolution are zero and ignored. The two output save sections are abutted together and the 1023-point linear convolution is written to the user-specified file.

The command line options that are given in the program's source code are reproduced for convenience in Fig 3-3. The following command line,

CONV INFILE/I OUTFILE/O FILFILE/F/D

was used to convolve the unit-step data files shown in Fig 3-4a and b. Note that the filter file was deleted after the operation. The resulting data file is shown in Fig 3-4c.

```
Command line:
   CONV input/I [/D] output/O filter/F [/D]

   where "input","output" and "filter" are any legal RDOS filenames.

   The input, output and filter filenames can be typed in any order,
   however, the I switch should always be attached to the input
   file, the O switch should always be attached to the output file,
   and the F switch should always be attached to the filter file.

   The D switch can only be attached to the input and filter files,
   and deletes these files after the output file has been created.
```

Fig 3-3 Program CONV command line options

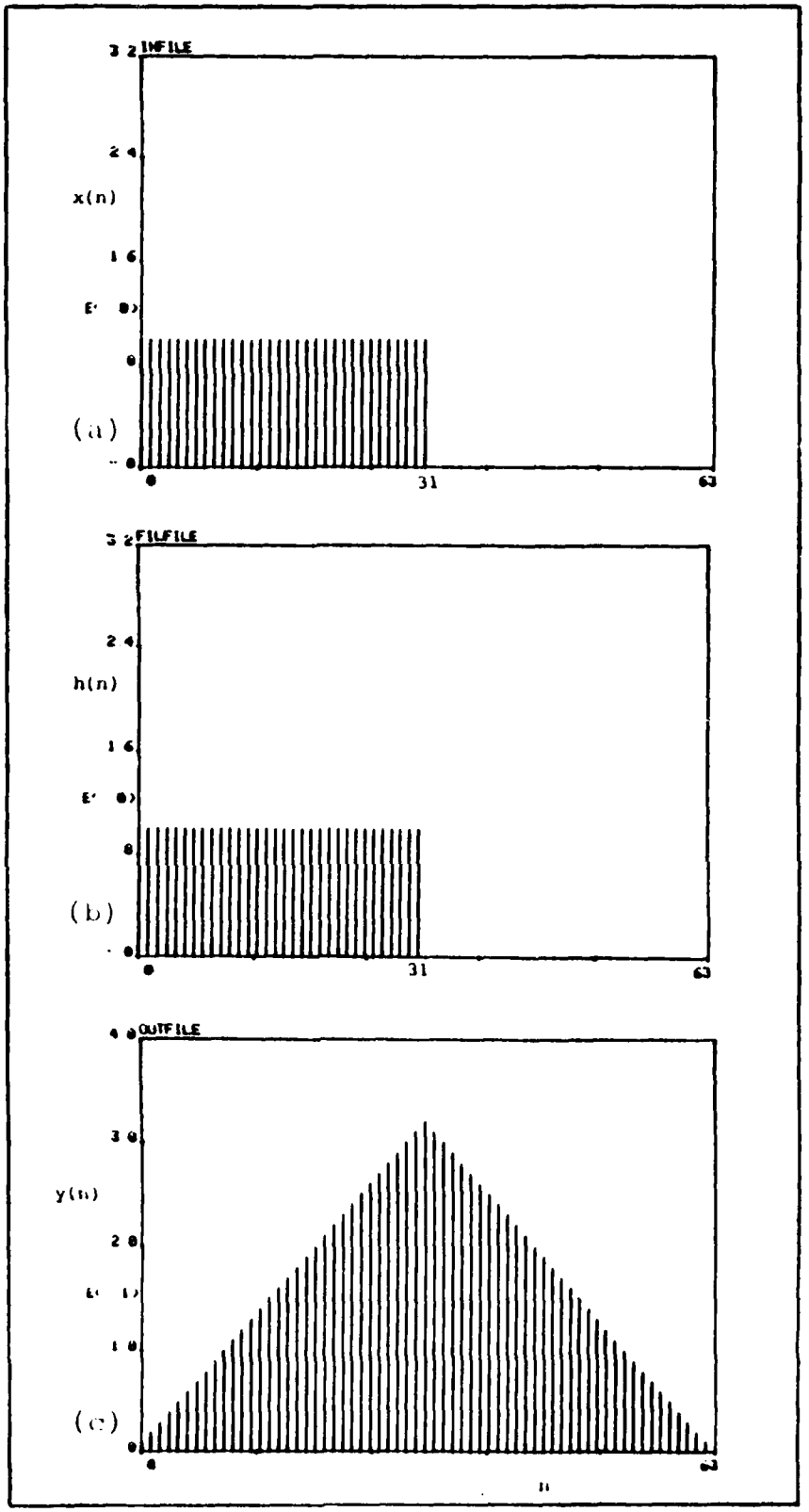


Fig 3-4 An example of using program CONV with two unit-step functions, (a) and (b), to obtain the linear convolution (c)

Programs for Frequency-Domain Processing

To allow signal processing in the frequency-domain, four DFT related programs, FFT, IFFT, MAG, and MULT, were generated that make use of the array processor. Program FFT computes the corresponding DFT of an input file on a 1024 or 2048-point basis. The input file must contain real number data and the output file will contain complex number data. If the input file contains less than the number of points required for the DFT, it is augmented with zeros as necessary. To complement this program, program IFFT computes the inverse DFT of a 1024 or 2048-point input file. The input and output file data types for this program are the reverse of that for the forward DFT program. Two other programs were created to operate on 1024 or 2048-point DFT data files. Program MAG computes the corresponding magnitude file of an input file and program MULT performs a point-by-point multiplication of two real or two complex input files. The multiplication program can thus be used to operate on both, complex number DFT data files and real number magnitude data files.

To compute the M-point DFT of real data on the array processor in a single DFT operation sequence, M/2 must be a power of 2 and within the limits of 8 and 1024. Three separate subroutine calls are required for the DFT operation sequence. Since the DFT result is complex, it requires twice the space of the time-domain real data. However, making use of symmetry properties, the array processor returns the DFT in a format that requires only half of this space (Ref 5:4-35).

The DFT result simply overwrites the original data.

The result of an M-point DFT is stored in array processor memory as follows. Since the first point and the M/2 point of the DFT are always real, these two points are stored in the first two points of the result. The following points form a complex array that contains the second through M/2-1 points of the DFT. To obtain the DFT for the M/2+1 through M points, this complex array is conjugated in reverse order.

To compute the inverse DFT, the DFT data file must be loaded into array processor memory in the same format as the forward DFT returns. The inverse DFT operation sequence can then be called to return the original time-domain real data.

The MULT and MAC programs use matrix operations to directly perform the specified operations on data. Subroutines VMCA and VMRA are used to perform a point-by-point multiplication of complex and real data, respectively. Subroutine VSMA is used to compute the square of the magnitude of complex data. The square root is taken prior to writing the results to file.

The process of multiplying two DFT data files of length N and M and then taking the inverse DFT is equivalent to performing a circular convolution. However, if the DFT data files were not created on the basis of at least an N+M-1 point DFT, the result will not be a linear convolution (Ref 6:111). Thus, when using this DFT package to implement a linear convolution of two data files, it must be remembered

that the sum of the two time-domain data file lengths cannot be larger than 1025 points when using the 1024-point DFT option or larger than 2049 points when using the 2048-point DFT option.

The command line format for each program is given in the program's source code. Each format is similar to that of the convolution program. Shown in Fig 3-5 is a 65-point discrete sine wave (a) and the corresponding magnitude of its DFT data file (b). The following command lines were used to obtain the magnitude data file,

```
FFT/S SINE/I DFTFILE/O
MAG/S DFTFILE/I DFTSINE/O
```

The S switch indicates that the 1024-point DFT option was used. The absence of this switch would implement the 2048-point DFT option. The following command line,

```
IFFT/S INVSINE/O DFTFILE/I
```

was used to retrieve the time-domain signal shown in Fig 3-5c. Only the first 65 points of the inverse DFT are shown since the coefficients are small beyond this length.

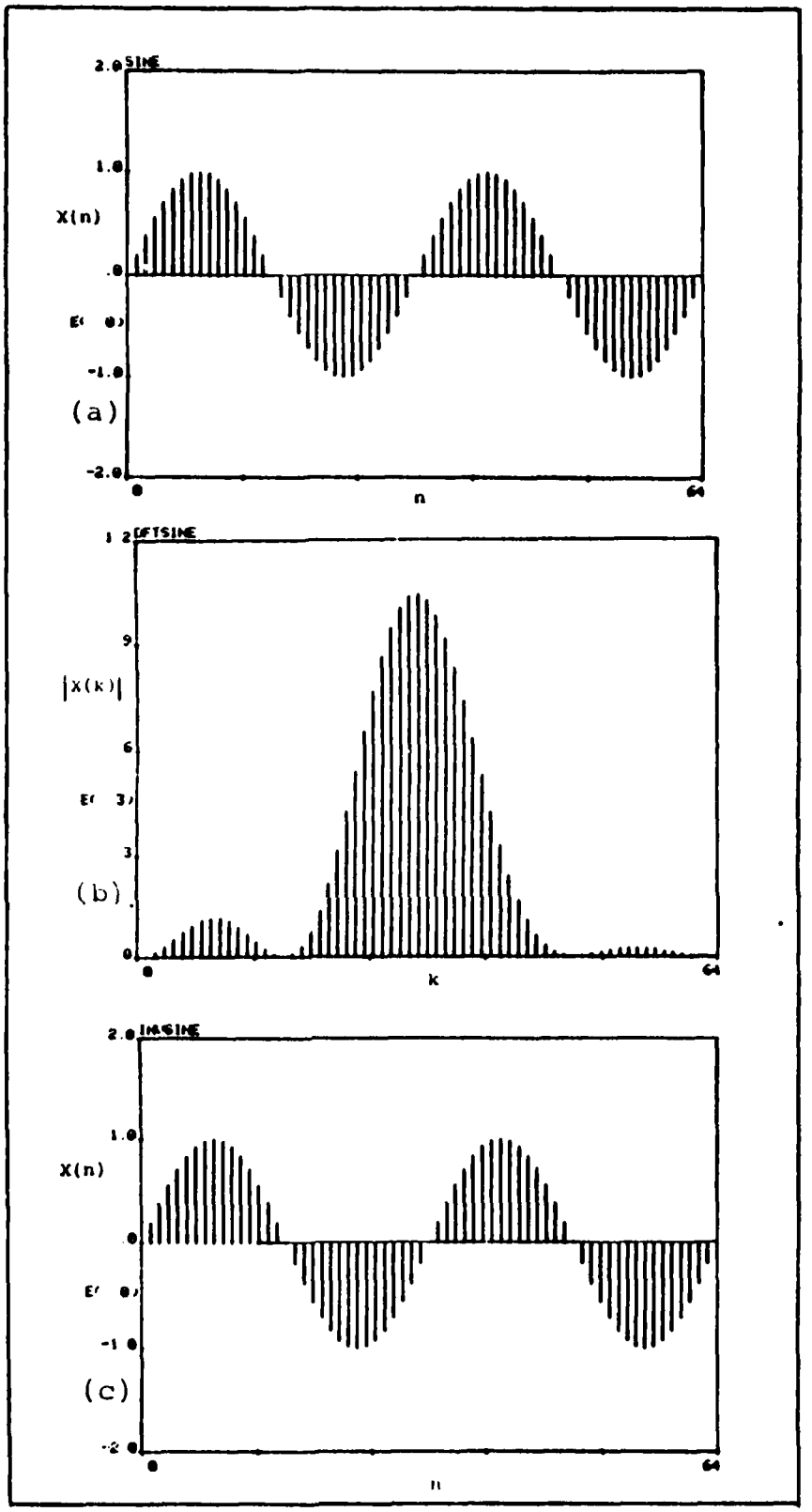


Fig 3-5 An example of DFT operations
(a) 65-point discrete sine wave
(b) the DFT magnitude obtained with program FFT and MAG
(c) the inverse DFT with program IFFT

IV Computer-Aided Design of Linear Phase FIR Filters

This chapter will present a brief description of the Parks-McClellan algorithm for designing linear phase FIR filters. The steps necessary to execute the IEEE machine-portable version (Ref 11:5.2-1) of this algorithm on the Eclipse will be given. The program, LPFIR, that was generated by modifying this algorithm will also be presented.

The Parks-McClellan Algorithm

Since the Parks-McClellan algorithm was presented in 1973 (Ref 7), it has appeared in textbooks (Ref 8:354-364; Ref 9:187-204) and been used in commercial software packages (Ref 10:18,27). The algorithm can be used to design a large class of linear phase FIR filters. It makes use of the Remez exchange, which is an efficient algorithm for designing digital filters with minimum weighted Chebyshev error.

The frequency response, H(f), of a FIR digital filter with a N-point impulse response, h(n), is the z-transform of the sequence evaluated on the unit circle. The frequency response of a linear phase filter can be written as,

$$H(f) = G(f) \exp[\, j2\pi f(L\pi/2-2(N-1))\,]$$

where G(f) is a real function and L=0 (for positive symmetry) or 1 (for negative symmetry). It can be shown that there are exactly four cases of linear phase FIR filters. These cases differ in length of the impulse response (even or odd) and the symmetry of the impulse response (positive or negative).

Positive symmetry is defined as $h(n) = h(N-1-n)$ and negative symmetry as $h(n) = -h(N-1-n)$.

In all four cases, a function $G(f)$ can be used to approximate the desired magnitude specification. Using symmetry relations, $G(f)$ can be expressed as follows for the four different cases. In all cases, $n = 1, 2, 3, \ldots k$.

Case 1: positive symmetry, odd length

$$G(f) = 2\sum_{n=0}^{k} h(k-n)\cos(2\pi f)$$

where $k = (N-1)/2$ and $h(k) = 1/2$

Case 2: positive symmetry, even length

$$G(f) = 2\sum_{n=1}^{k} h(k-n)\cos(2\pi f(n-1/2))$$

where $k = n/2$

Case 3: negative symmetry, odd length

$$G(f) = 2\sum_{n=1}^{k} h(k-n)\sin(2\pi nf)$$

where $k = (N-1)/2$ and $h(k) = 0$

Case 4: negative symmetry, even length

$$G(f) = 2\sum_{n=1}^{k} h(k-n)\sin(2\pi f(n-1/2))$$

where $k = N/2$

Earlier efforts at designing FIR filters concentrated on Case 1. But Parks and McClellan presented a method of combining all four cases into one algorithm. This was done by using symmetry relations to express the other three cases

as a form of the first case, Q(f), multiplied by a function, P(f). This allows all four cases to be expressed as $G(f) = Q(f)P(f)$, where P(f) is a linear combination of cosine functions. Since all four cases can be expressed in a common form, a single computation routine (the Remez exchange) can be used to calculate the filter approximation.

The filter approximation is obtained as follows. Given a desired magnitude response, D(f), and a positive weight function, W(f), both continuous over a compact subset $F \subset [0, \frac{1}{2}]$, the absolute weighted error is defined as,

$$||E(f)|| = \max_{f \in F} W(f) |D(f)-G(f)|$$

Defining the frequency domain, F, in this manner implies a sampling rate of 1. The above expression can be rewritten as,

$$||E(f)|| = \max_{f \in F} W(f)Q(f) |D(f)/Q(f)-P(f)|$$

This expression can be used to calculate the best approximation based only on cosine functions. The minimum weighted error can be obtained by careful choice of the coefficients of P(f). The alternation theorem is used to determine the number of cosine functions necessary. By making use of the error function and the conditions of the alternation theorem, the Parks-McClellan algorithm provides the best filter approximation.

Implementation on the Eclipse

The referenced IEEE publication contained the source code for the Parks-McClellan algorithm used in program LPFIR. This publication was the result of an effort to collect and make some of the popular DSP programs more available and machine independent. In the IEEE version, the Parks-McClellan algorithm had several variations from the original paper. The major difference was with the REMEZ subroutine. This subroutine was changed significantly to allow it to use variables already present in a common block to compute the cosine functions, instead of passing additional variables as arguments. Also, throughout the entire software package, minor changes were made that affected mixed-mode arithmetic and library subroutines.

The only system dependent parameters that had to be defined for the IEEE version were the I/O unit numbers. The READ statements were changed to ACCEPT statements to allow data to be easily input from the console. After these changes were made, the source code for the mainline and subroutines were separately compiled and the program was loaded. The program was executed without any problems.

Each filter example given in the IEEE publication and the original paper was reproduced with the program on the Eclipse. The impulse response for all examples was identical within 5-6 decimal positions. The program output given on page 5.1-7 of the IEEE publication for a 55-point multiband filter is given as Fig 4-1. The filter design with the pro-

gram executed on the Eclipse for the same parameters is given as Fig 4-2.

Program Description

The Parks-McClellan algorithm as listed in the IEEE publication and other referenced sources was intended for use with a card reader. To allow data input to be more descriptive, this section of code was modified to request the parameters from the user and give appropriate ranges of values. The program was also modified to be executed in a command line format similar to that of the signal processing programs. The command line options that are given in the program's source code are reproduced for convenience in Fig 4-3.

There are several options available for executing the program. A parameter file is created for each set of filter parameters given to the program. This file can be optionally deleted after designing the filter impulse response. It can also be retained and the next filter design made by simply referencing the existing parameter file. Each filter design is also written to a file. The filter output listing, such as that given in Figs 4-1 and 2, can be optionally printed. A new parameter file can be created and an existing parameter file can be viewed and/or altered without designing the corresponding filter. An example of the parameter file display that is given each time the program is executed is shown in Fig 4-4. This figure is the display

```
DEVIATION =  0.000734754
DEVIATION =  0.006315947
DEVIATION =  0.021567374
DEVIATION =  0.026203127
DEVIATION = -0.032680369
DEVIATION = -0.034435446
DEVIATION = -0.034448378
DEVIATION = -0.034448593

**********************************************************************

          FINITE IMPULSE RESPONSE (FIR)
        LINEAR PHASE DIGITAL FILTER DESIGN
            REMEZ EXCHANGE ALGORITHM
                BANDPASS FILTER
              FILTER LENGTH =  55
          ***** IMPULSE RESPONSE *****
        H( 1) =  0.10662652E-02 = H( 55)
        H( 2) =  0.63777615E-02 = H( 54)
        H( 3) =  0.35755609E-02 = H( 53)
        H( 4) = -0.90677854E-02 = H( 52)
        H( 5) = -0.90906978E-02 = H( 51)
        H( 6) =  0.29155630E-02 = H( 50)
        H( 7) =  0.39637965E-02 = H( 49)
        H( 8) =  0.11172051E-01 = H( 48)
        H( 9) =  0.11646759E-01 = H( 47)
        H(10) = -0.99630785E-02 = H( 46)
        H(11) = -0.92384245E-02 = H( 45)
        H(12) = -0.20406392E-01 = H( 44)
        H(13) = -0.19460483E-01 = H( 43)
        H(14) =  0.31243014E-01 = H( 42)
        H(15) =  0.63045568E-02 = H( 41)
        H(16) = -0.20482803E-01 = H( 40)
        H(17) =  0.65740513E-02 = H( 39)
        H(18) = -0.11202127E-02 = H( 38)
        H(19) =  0.41956986E-01 = H( 37)
        H(20) =  0.35784266E-01 = H( 36)
        H(21) =  0.34744803E-01 = H( 35)
        H(22) =  0.71496359E-01 = H( 34)
        H(23) = -0.17138831E 00 = H( 33)
        H(24) = -0.18255044E 00 = H( 32)
        H(25) =  0.74059024E-01 = H( 31)
        H(26) = -0.10317421E 00 = H( 30)
        H(27) =  0.25716721E-01 = H( 29)
        H(28) =  0.37813546E 00 = H( 28)

                      BAND  1        BAND  2        BAND  3        BAND  4
LOWER BAND EDGE        0.            0.1000000      0.1800000      0.3000000
UPPER BAND EDGE        0.0500000     0.1500000      0.2500000      0.3600000
DESIRED VALUE          0.            1.0000000      0.             1.0000000
WEIGHTING             10.0000000     1.0000000      3.0000000      1.0000000
DEVIATION              0.0034449     0.0344486      0.0114829      0.0344486
DEVIATION IN DB      -49.2565703     0.2941783    -38.7989955      0.2941783

                      BAND  5
LOWER BAND EDGE        0.4100000
UPPER BAND EDGE        0.5000000
DESIRED VALUE          0.
WEIGHTING             20.0000000
DEVIATION              0.0017224
DEVIATION IN DB      -55.2771702

EXTREMAL FREQUENCIES--MAXIMA OF THE ERROR CURVE
    0.          0.0167411   0.0323661   0.0446429   0.0500000
    0.1000000   0.1089286   0.1267857   0.1424107   0.1500000
    0.1800000   0.1855804   0.1978571   0.2134821   0.2302232
    0.2436160   0.2500000   0.3000000   0.3122768   0.3323661
    0.3502232   0.3600000   0.4100000   0.4155804   0.4289732
    0.4457143   0.4635714   0.4814285   0.5000000

**********************************************************************
```

Fig 4-1 Sample program output from IEEE publication

```
DEVIATION =    .000734925
DEVIATION =    .006317233
DEVIATION =    .021567347
DEVIATION =    .026203358
DEVIATION =   -.032680656
DEVIATION =   -.034435285
DEVIATION =   -.034448225
DEVIATION =   -.034448442

********************************************************************************

                FINITE IMPULSE RESPONSE (FIR)
            LINEAR PHASE DIGITAL FILTER DESIGN
                REMEZ EXCHANGE ALGORITHM

                    BANDPASS FILTER

                  FILTER LENGTH =  55

              ***** IMPULSE RESPONSE *****
            H( 1) =    .10662480E-02 = H( 55)
            H( 2) =    .63777040E-02 = H( 54)
            H( 3) =    .35756380E-02 = H( 53)
            H( 4) =   -.90677220E-02 = H( 52)
            H( 5) =   -.90907510E-02 = H( 51)
            H( 6) =    .29155130E-02 = H( 50)
            H( 7) =    .39637950E-02 = H( 49)
            H( 8) =    .11172020E-01 = H( 48)
            H( 9) =    .11646760E-01 = H( 47)
            H(10) =   -.99629980E-02 = H( 46)
            H(11) =   -.92383360E-02 = H( 45)
            H(12) =   -.20406370E-01 = H( 44)
            H(13) =   -.19460540E-01 = H( 43)
            H(14) =    .31242970E-01 = H( 42)
            H(15) =    .63045660E-02 = H( 41)
            H(16) =   -.20482860E-01 = H( 40)
            H(17) =    .65740350E-02 = H( 39)
            H(18) =   -.11202380E-02 = H( 38)
            H(19) =    .41956930E-01 = H( 37)
            H(20) =    .35784270E-01 = H( 36)
            H(21) =    .34744800E-01 = H( 35)
            H(22) =    .71496550E-01 = H( 34)
            H(23) =   -.17138810E 00 = H( 33)
            H(24) =   -.18255050E 00 = H( 32)
            H(25) =    .74059010E-01 = H( 31)
            H(26) =   -.10317420E 00 = H( 30)
            H(27) =    .25716610E-01 = H( 29)
            H(28) =    .37813530E 00 = H( 28)

                          BAND  1       BAND  2       BAND  3       BAND  4
LOWER BAND EDGE          .0000000      .1000000      .1800000      .3000000
UPPER BAND EDGE          .0500000      .1500000      .2500000      .3600000
DESIRED VALUE            .0000000     1.0000000      .0000000     1.0000000
WEIGHTING              10.0000000     1.0000000     3.0000000     1.0000000
DEVIATION                .0034448      .0344484      .0114828      .0344484
DEVIATION IN DB        -49.2563900      .2941704   -38.7990100      .2941704

                          BAND  5
LOWER BAND EDGE          .4100000
UPPER BAND EDGE          .5000000
DESIRED VALUE            .0000000
WEIGHTING              20.0000000
DEVIATION                .0017224
DEVIATION IN DB        -55.2771900

EXTREMAL FREQUENCIES--MAXIMA OF THE ERROR CURVE
      .0000000      .0167411      .0323661      .0446428      .0500000
      .1000000      .1089283      .1267849      .1424094      .1500000
      .1800000      .1855802      .1978566      .2134811      .2302217
      .2436141      .2500000      .3000000      .3122764      .3323651
      .3502217      .3600000      .4100000      .4155802      .4289727
      .4457132      .4633698      .4814264      .5000000

********************************************************************************
```

Fig 4-2 Sample program output from Eclipse

```
Command line:
   LPFIR parameter/P [/E] [/D] [filter/F [/L] ]

   where "parameter" and "filter" are any legal RDOS filename

   The P switch must always be attached to the parameter filename. A
   parameter file will be created with the filter parameters
   interactively specified by the user. The filter parameters will
   be displayed and can be changed if requested by the user.

   The E switch denotes that the parameter file already exists. The
   filter parameters will be display and can be changed if requested
   by the user.

   The filter filename and F switch denotes that the filter specified
   by the parameter file will be designed and the impulse response
   stored under the filter filename. The F switch must be attached.

   The L switch denotes that a listing for the filter design will
   be sent to the printer.

   If the parameter and filter files are both given, they can be
   typed in any order.

   The D switch can only be attached to the parameter file if a
   filter file is also specified. This switch deletes the parameter
   file after the filter file has been created.
```

Fig 4-3 Program LPFIR command line options

```
              --) Multiple Passband/Stopband Filter (--
Parameter File: PFILE                          Filter File: not specified
Filter Length:  55   Number of Bands:  5       Grid Density: 16

                        Lower        Upper       Frequency      Weight
                        Cutoff       Cutoff      Response       Function

Band Number  1          .0000        .0500          0.            10.
Band Number  2          .1000        .1500          1.             1.
Band Number  3          .1800        .2500          0.             3.
Band Number  4          .3000        .3600          1.             1.
Band Number  5          .4100        .5000          0.            20.




Do you want to,
   1: accept the above parameters
   2: change the above parameters
selection:
```

Fig 4-4 Program LPFIR parameter file display

for the parameter file used to generate the output listing in Fig 4-2.

The program fails to design many filters with reasonable design parameters. Also, design parameters that yield a well-designed filter can be changed only slightly and will yield a very poorly designed filter. The Parks-McClellan algorithm returns an error message if the REMEZ routine fails to find a proper set of cosine functions to approximate the filter. However, the algorithm does not give any error messages for poorly designed filters, that is, filters with a large amount of ripple and with a frequency response that differs greatly from the desired amount. All filter designs from the program should be verified with a DFT prior to use.

A systematic approach was found to allow the filter design parameters that most closely approach the desired filter to be found within several filter design iterations. This method is presented in the form of a user's manual for program LPFIR. Basically, the user begins with a design far less stringent than what is desired. Parameters are separately adjusted until further adjustment does not yield a better filter design. The user's manual and the source code for the mainline and subroutines of program LPFIR are given in Appendix E. The user's manual also explains how to set up a macro file that contains programs LPFIR, FFT, MAG, and FILTPLOT. This macro allows the user to design and display filters quickly in an interactive environment.

V Conclusion

This chapter will summarize the results of this effort and give three recommendations of how the signal processing system could be improved.

Summary

The purpose of this effort was to increase the capability of the AFIT signal processing laboratory and to make it more user-oriented. Three areas--digitizing operations, signal processing operations, and digital filter design were considered for expansion. Software was generated that made use of the Eclipse A/D/A device's two main features, the capability to work with large data files and have conversion operation options interactively set by the user. A user's manual for this device, that is intended to replace the Data General documentation, was written to aid in writing future software. The array processor was utilized in several signal processing programs that are executed by typing a single line command. A convolution program was generated that allows large data files to be convolved with filters containing up to 512 points. Programs were also generated to allow Fourier Transform related operations to be performed on data files containing up to 2048 points. An existing filter design program, capable of building a wide variety of linear phase FIR digital filters, was modified to allow easy operation and execution on the Eclipse. A user's manual for this program was written to give guidance in adjusting the filter design

parameters to obtain the desired digital filter. This software package will allow personnel using the laboratory to perform additional signal processing operations.

Recommendations

Applying the Eclipse A/D/A device to perform video digitizing operations should be investigated. This would allow the laboratory to have a back-up digitizing capability in both, the speech and video areas.

The array processor will not be utilized by the majority of users in the laboratory until the degree of difficulty in operating this device is reduced. A way this could be done is by creating corresponding stand-alone subroutines for each of the array processor matrix operations. This would free the user from dealing with array processor memory, setting up matrix operation control blocks, and arranging data in peculiar formats required by some operations. All of the matrix operations deal with small amounts of data, since array processor memory is only 8KW. Data could be interchanged between the mainline and subroutine as arguments and array processor memory transfers could be handled with the VSTR and VLDR subroutines. It would require more overhead to perform each matrix operation, however, the speed of the array processor and the increase in its usage would make this effort worthwhile.

The filter design program should be revised to allow filters containing up to 512 points to be built, since the convolution program can handle filters of this length. The

current filter design program has a filter length limit of 256 points. The following variables within this program can be adjusted, however, as shown below to allow filters of any size to be built.

Name	Dimension
IEXT	MAX/2+2
AD	MAX/2+2
ALPHA	MAX/2+2
X	MAX/2+2
Y	MAX/2+2
H	MAX/2+2
WT	16(MAX/2+2)
DES	16(MAX/2+2)
GRID	16(MAX/2+2)

where MAX = the maximum filter length

Revising the current program to build 512-point filters would require doubling the space of the variables given above from 14.5KW to 29.0KW. This would cause the program's executable save file to exceed the 32KW maximum.

An approach to solving this problem would be to use extended memory to hold two of the large data arrays, DES and GRID. By placing these arrays in extended memory, the mainline can be reduced to an acceptable size. The array elements can be accessed by the mainline and subroutines by using the VSTASH and VFETCH calls. Since these are real arrays, the following lines of code could be placed in the mainline to set up the extended memory window,

```
REAL WIND(1024)
CALL MAPDF(17,WIND,1,2,IER)
```

The data array subscripts could be changed as shown on the next page to place DES at the top of extended memory,

followed by GRID.

Array	Original Subscript	Revised Subscript
DES	J	J
GRID	K	K+4128

An example of how to revise the source code to allow data to be transferred between the mainline and extended memory is shown below,

Original Code	Revised Code
GRID(K) = DELF+NFCNS	HOLD = DELF+NFCNS PLACE = K+4128 CALL VSTASH(HOLD,PLACE)
H(I) = GRID(K)+DELF	PLACE = K+4128 CALL VFETCH(HOLD,PLACE) H(I) = HOLD+DELF

where HOLD is a real variable
PLACE is an integer variable

The AFIT signal processing laboratory has grown tremendously in the past few years and plans have been made for additional expansion. This is an indication of the notable research that the laboratory is used to support. Although the research is directed toward military application, many civilian areas would also benefit. It is hoped that this effort will aid future research by allowing personnel to make better use of the laboratory's capability.

Bibliography

1. Data General Corporation. Analog Data Subsystem for NOVA and ECLIPSE Line Computers Models 4330-4333. Programmer's Reference Series, 014-000651, November 1980.

2. Data General Corporation. Analog Data Subsystem Models 4330-4331. Technical Reference Series, 014-000652, November 1980.

3. Data General Corporation. Sensory Access Manager User's Manual. 093-00025-02, August 1980.

4. Data General Corporation. Eclipse-Line Real Time Disk Operating System Reference Manual. 093-000129-01, September 1975.

5. Data General Corporation. Array Processor Software User's Manual. 093-000169-00, October 1978.

6. Oppenheim, Alan V. and Ronald W. Schafer. Digital Signal Processing. Englewood Cliffs: Prentice-Hall Inc., 1975.

7. McClellan, James H., Thomas W. Parks, and Lawrence R. Rabiner. "A Computer Program for Designing Optimum FIR Linear Phase Digital Filters," IEEE Transactions on Audio and Electroacoustics, 21 (6);506-525 (December 1973).

8. Cappellini, V., A. G. Constantinides, and P. Emiliani. Digital Filters and Their Applications. New York: Academic Press Inc., 1978.

9. Rabiner, Lawrence R. and Bernard Gold. Theory and Application of Digital Signal Processing. Englewood Cliffs: Prentice-Hall Inc., 1975.

10. Signal Technology, Inc. Digital Filtering. Interactive Laboratory System (ILS) software user's manual for filtering package.

11. IEEE. Programs for Digital Signal Processing. Edited by the Digital Signal Processing Committee, IEEE Acoustics, Speech, and Signal Processing Society. New York: IEEE Press, 1979.

Appendix A

The Eclipse A/D/A Device User's Manual

Air Force Institute of Technology

Department of Electrical Engineering

Digital Signal Processing Laboratory

Eclipse A/D/A Device

User's Manual

Original Release

Dec 82

Preface

The Eclipse computer in the AFIT signal processing laboratory is equipped with a model 4331 analog data subsystem and the Sensory Access Manager, SAM, software package. The SAM software package aids in building I/O programs for Data General computers equipped with appropriate analog-to-digital-to-analog devices, such as the model 4331 subsystem.

This manual explains how to write application programs that operate the Eclipse A/D/A device and concentrates on methods of most interest for signal processing applications. To focus in this area, the scope of this manual will be limited to working with SAM in Fortran V and DG assembly language and operating the model 4331 subsystem in data channel mode. The model 4331 subsystem and SAM, however, are versatile and have other features that will only be mentioned. The SAM User's Manual and the Models 4330-4333 Programmer's and Technical References, all written by Data General, should be consulted for detailed descriptions of these additional features.

All sample programs included in this manual have been verified to operate on the Eclipse computer in the <u>exact</u> format shown.

Contents

List of Figures

List of Tables

Chapter 1
Introduction to SAM

Call Categories

The Sensory Access Manager, SAM, is a software package that simplifies the building of I/O programs which utilize A/D/A devices. SAM allows these devices to be operated flexibly through the use of Fortran IV, Fortran V, or assembly language calls. This manual will only discuss usage of Fortran V and assembly language calls, since Fortran IV programming is usually not used on the Eclipse computer. An application program can perform A/D conversions or D/A conversions or both. A single conversion operation, however, can only perform A/D or D/A conversions.

The conversion calls can post either a single-operation or multiple-operation request. A single-operation request specifies a cyclist list of channels from which data will be collected or sent, a total conversion count for all channels, and a single clock source which is used to trigger conversions. Multiple-operation requests set up a series of single-operation requests. With both types of requests, program control can either be suspended until the request defined has been completed or returned immediately, in which case the program must check for completion later. Since the main advantage of multiple-operation requests is the ability to operate more than one device, their usage will not be discussed in this manual.

The Eclipse A/D/A device can be operated in one of two modes according to how it will move conversion data--programmed I/O or data channel I/O. With programmed I/O, data is moved through an accumulator where it is readily available to the program for manipulation. However, because one or more instructions must be executed for each word transferred, programmed I/O is slow and generally used only when small quantities of information are transferred. Data channel I/O reduces the amount of program overhead by transferring blocks of data automatically via the data channel. Once the data channel transfer for a block of data has been set up and initiated by the program, no further action by the program is required to complete the transfer. This is the only method of data transfer that will be discussed in this manual.

Programming Trade-offs

The manner in which the Fortran and assembly language calls interface with an application program to control a device is shown in Fig 1-1. Fortran interfaces work through the assembly level interface. The assembly level interface works through the operating system which drives the devices. Since the assembly language calls require less overhead, they are faster and allow more space in the main program for data storage. If the user is familiar with assembly language programming, operating the device at this level is not much more difficult than operating the device with Fortran programming. This is because the assembly language macros provided in the

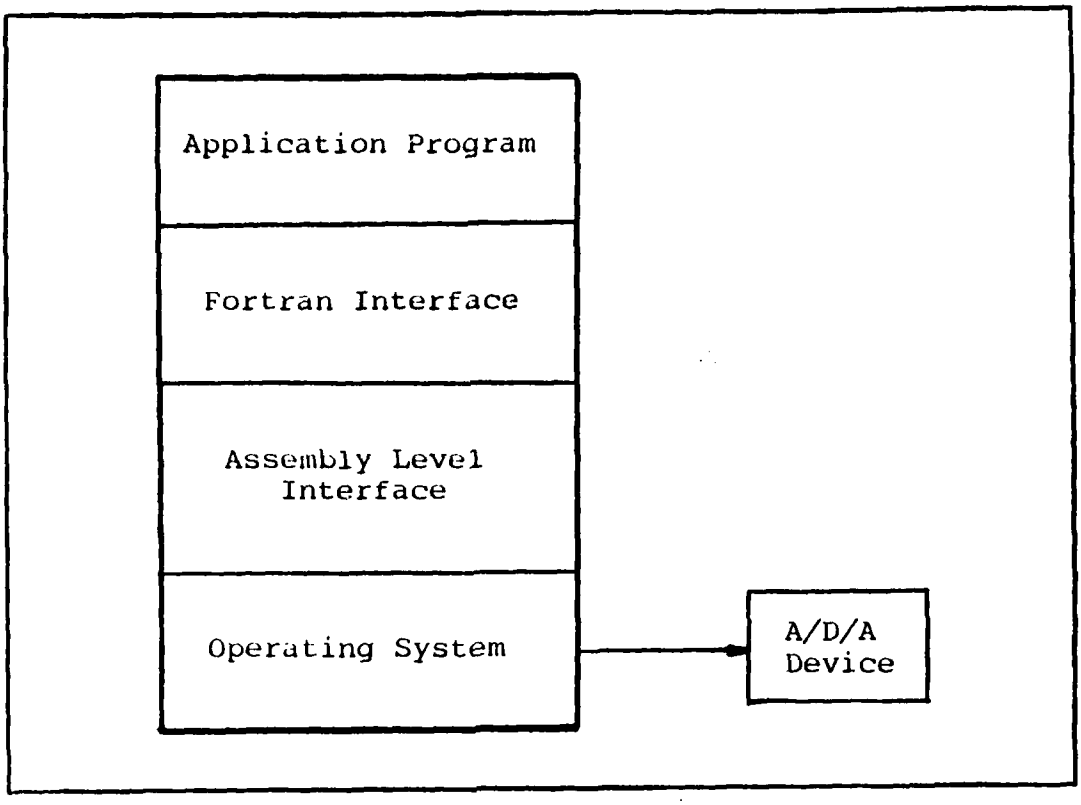


Fig 1-1 SAM Interfaces

SAM libraries allow the device to be operated in a manner similar to that of Fortran programming. With Data General Fortran V programming, it is possible to write the main program for data manipulation in Fortran V and then call upon a subroutine, overlay, or swap written in assembly language to operate the device. This method would be useful, for instance, when repeated A/D conversion calls would be necessary to collect the desired number of data samples. Writing this section of code in assembly language would allow data to be moved from the data buffer more quickly, thus losing fewer data samples between repeated A/D conversion calls. If the

user is not familiar with assembly language programming, however, operating the device at the Fortran level will be much easier. The Fortran calls can perform any single conversion operation request that can be done with assembly language calls. If repeated conversion calls are not necessary to handle the desired amount of data, then nothing significant is gained by using assembly language calls. This is because once conversion operations begin triggering, Fortran calls and assembly language calls are handled by the operating system in the same manner. If the main program cannot provide enough space for the Fortran overhead and data arrays, then a program swap or overlay can be used to handle the conversion operation. The largest number of data samples any single conversion operation can handle is 16,384. The Fortran overhead and data arrays to handle such a single conversion operation will fit within the maximum length of 32KW that can be allotted for an executable overlay or swap file. Basically, program swaps operate by overwriting main memory with a new program, while overlays overwrite a section of main memory with new code. Program swaps are easier to learn to use and set up than overlays. Therefore, if a secondary file will be used to handle the conversion operation and the file is near 32KW long or if the additional processing delay caused by program swaps can be tolerated, then a program swap would be the better method. An example of a program swap setup is given in Fig 1-2.

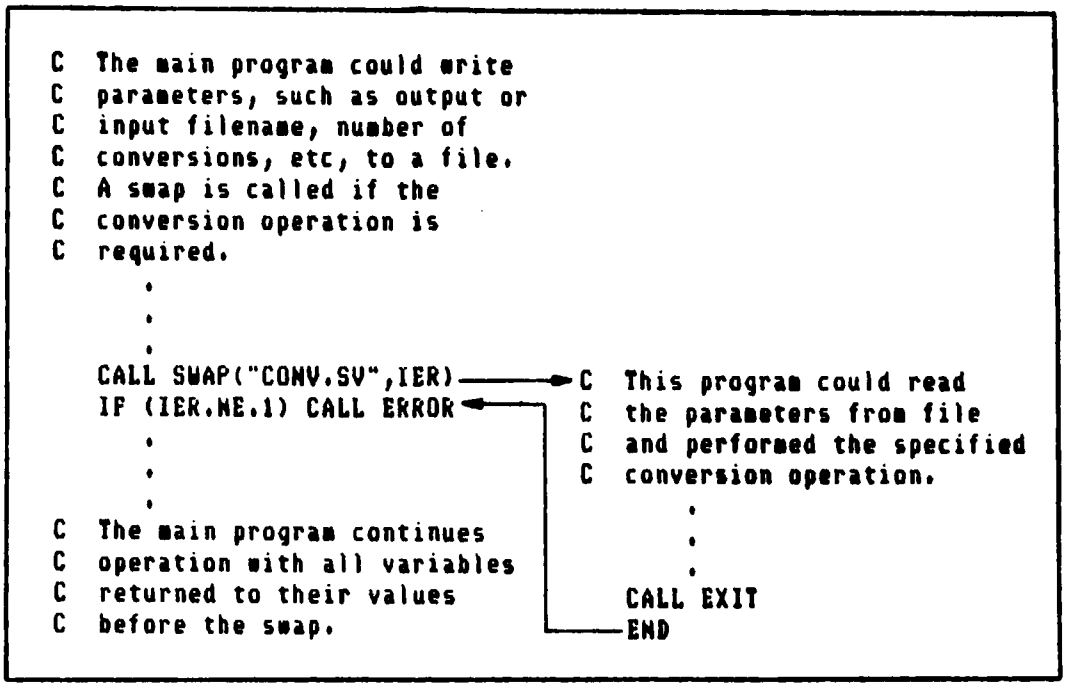


Fig 1-2 Program SWAP setup

Operating Overview

The basic additions that must be included in an application program to perform A/D/A operations are quite simple. First, the device op-codes and conversion data buffers, specified by the configuration file to be used, are declared at the beginning of the program. Second, the operating system is initialized with a SAM library subroutine call. Finally, after setting the arguments to appropriate values, another SAM library subroutine is called to perform the operation.

The SAMGEN program located in the SAM directory of DP4F is used to create the configuration file. The configuration file defines parameters used by the operating system to operate the I/O device. It is loaded with the main program in the RLDR command line. All conversion operations in the application program must adhere to the framework set up by the configuration file loaded with the program.

This manual is divided into chapters which discuss each of the steps necessary to run an A/D/A application program. Chapter 2 describes the basic capabilities of the Eclipse A/D/A device and how to set the argument values for the Fortran and assembly language conversion operation calls. Chapter 3 describes how to build configuration files. The general program setup and the different SAM subroutine calls are described in Chapter 4 for Fortran V programming and Chapter 5 for assembly language programming. Finally, Chapter 6 describes how to compile and load application programs.

Chapter 2
The Model 4331 Subsystem

General Information

The model 4331 analog data subsystem is a stand-alone device which contains both the A/D and D/A converters. It consists of a 15-inch printed circuit board that fits in a slot on the Eclipse computer chassis. The Eclipse computer contains many such slots for expansion. User interface to the circuit board is provided through two connector paddleboards--one called analog and one called digital. The pin connections for these two paddleboards and other specifications are given in the Models 4330-4333 Technical Reference.

The subsystem contains independent software interfaces for A/D and D/A operations. Each interface has its own device code and must be accessed separately in the application program. The A/D section is organized around a single 12-bit A/D converter and two multiplexors. The multiplexors allow input of up to 16 differential signals. The D/A section is organized around two 12-bit D/A converters. The A/D and D/A converters can be set with jumpers to operate at a voltage range of 0 to 5v, 0 to 10v, ±5v, or ±10v with conversion values returned in either straight binary or two's complement format. Both converters have been set to operate at the ±5v range in two's complement format.

A conversion value is stored in one machine word with bit 0 used as a sign bit and bits 1-11 used to store the

value. The remaining bits, 12-15, are always returned zero from an A/D operation and are ignored on a D/A operation. The bit settings are shown in Fig 2-1 for the most positive conversion value, 077760K, and the most negative conversion value, 100000K.*

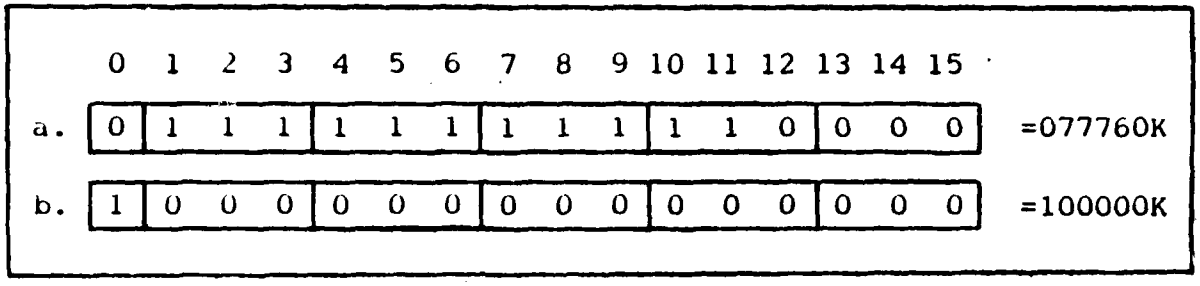


Fig 2-1 Conversion value stored for (a) most positive and (b) most negative value

The 12 bits provide 4,096 different conversion values, with 2,048 allocated for negative values, 2,047 allocated for positive values, and one allocated for the value zero. Since the device is set for a full range of 10 volts, each increment represents approximately .0024 volts. Shown in Table 2.1 is a list of stored values and corresponding voltages for several conversion numbers. Since integer numbers on the Eclipse computer also occupy one machine word using the same two's complement format, a one-to-one correspondence can be made between a sampled value and its integer value. Due to the

* The symbol K indicates octal format. This notation is adapted from that used in Data General Fortran V to indicate that a number is in base eight representation.

least four significant bits of each word not being used, conversion values occur in corresponding integer-value multiples of sixteen. Making use of the integer-value correspondence, a conversion value can be changed to its actual voltage value with a single line of Fortran code such as the following,

```
REALNUM=FLOAT(INTNUM)/32768.*5.
```

where INTNUM is the one-word conversion value and REALNUM is the corresponding two-word, real number representation of the actual sampled value.

To request a conversion operation, variables are passed via Fortran or assembly language calls. The following section explains how to set the variable values for each type of call. The IDATAx words indicate Fortran variables and the CDATx words indicate assembly language variables.

Conversion Number	Octal Value Stored	Integer Value Stored	Actual Sampled Value
-2048	100000	-32768	-5.0000v
-2047	100020	-32752	-4.9976v
-2046	100040	-32736	-4.9951v
-1536	120000	-24576	-3.7500v
-512	160000	-8192	-1.2500v
-2	177740	-32	-0.0049v
-1	177760	-16	-0.0024v
0	000000	0	0.0000v
1	000020	16	0.0024v
2	000040	32	0.0049v
512	020000	8192	1.2500v
1536	060000	24576	3.7500v
2046	077740	32736	4.9951v
2047	077760	32752	4.9976v

Table 2.1 Conversion values stored and corresponding voltages

Variable Definitions

The variables passed for both A/D and D/A requests indicate the channel use numbers, conversion count, clock source, and the storage location for conversion values.

The channel use numbers are specified differently for A/D or D/A operations. For A/D operations, the initial and final channel numbers are specified. The A/D channels are numbered 0-15. For example, if 4 was given as the start channel and 7 as the final channel, conversions would be taken from channels 4, 5, 6, 7, 4, 5, 6, etc. The converter will wrap around from channel 15 to channel 0. For example, if 13 and 2 were given as the start and final channels, conversions would be taken from channels 13, 14, 15, 0, 1, 2, etc. To specify a fixed channel, the same value is entered for both the initial and final channel. For D/A operations, the initial channel and fixed/alternate mode are specified. The D/A channels are numbered 0 and 1.

The device offers four clock sources--pulse, DCH, internal, and external. The A/D converter can use all four clocks, however, the D/A converter can use only the pulse, internal, and external clocks. The pulse clock triggers conversions from software generated pulses and the DCH clock triggers conversions at the maximum rate the device allows. These two clocks are not as useful for signal processing applications as are the internal and external clocks. It is more difficult to control the pulse clock rate with precision than it is for the external clock. The DCH clock rate is too

fast for most signal processing applications. Using a TTL pulse generator as an external clock, with a frequency counter to measure the clock period, the external clock provides an accurate, versatile clock source. The A/D and D/A converters have separate connections for external clocks. The internal clock can be set for a clock period range of 45-300 microsec. However, the adjusting mechanism for this clock is a screwdriver, variable resistor on the main printed circuit board. Since this board must be removed from the Eclipse computer for clock adjustment, the external clock must be used if the current internal clock setting is not what is desired. Currently, the internal clock is set for a clock period of 46 microsec.

The variable IDATA1/CDAT1 occupies one machine word and specifies the clock source and the channel use numbers. In Fortran, IDATA1 can be an integer or an integer variable. In assembly language, CDAT1 is the variable value. The bit definitions of IDATA1/CDAT1 are shown in Table 2.2 for an A/D operation and in Table 2.3 for a D/A operation. The bit settings for the clock source with either type of operation are shown in Table 2.4. Two points should be remembered when setting these bits. First, it is illegal to set the bits for DCH clock on a D/A operation. Second, use of the pulse clock requires additional software and should only be attempted after consulting the SAM User's Manual (p. 4-28) for setup. Use of the pulse clock does not affect the type

Bit Numbers	Function
0	(ignored)
1-2	clock source
3	fixed/alternate mode
4	(set to one)
5-14	(ignored)
15	start channel

Table 2.2 Variable IDATA1/CDAT1 bit definitions for A/D operation

Bit Numbers	Function
0	(ignored)
1-2	clock source
3-5	(set to zero)
6-9	final channel
10-11	(set to zero)
12-15	start channel

Table 2.3 Variable IDATA1/CDAT1 bit definitions for D/A operation

Bit 1	Bit 2	Clock Selected
0	0	pulse
0	1	DCH
1	0	internal
1	1	external

Table 2.4 Variable IDATA1/CDAT1 clock source bit settings

of SAMGEN configuration file used. The user, however, must generate an assembly language module to trigger conversions. This module is called in the application program for each conversion triggered. The bit settings for channel use numbers are shown in Table 2.5 for an A/D operation and in Table 2.6 for a D/A operation.

A convenient method for setting the bit values with Fortran operation is to use the .OR. operator. The octal values that correspond to setting the required bits for different options are shown in Table 2.7 for an A/D operation and in Table 2.8 for a D/A operation. In addition to setting the bits for clock operation, the clock values shown also set the miscellaneous bits. The following line of code could be used to set IDATA1 to collect sampled data on channel 10 using external clock,

```
IDATA1=(60000K.OR.1000K).OR.10K
```

The following line of code could be used to set IDATA1 to output conversion data on channel 1 using internal clock,

```
IDATA1=44000K.OR.1K
```

For D/A operation, if alternate mode is not specified, fixed mode is assumed.

Start	Bit 12	Bit 13	Bit 14	Bit 15	Channel Selected
Final	Bit 6	Bit 7	Bit 8	Bit 9	
	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	8
	1	0	0	1	9
	1	0	1	0	10
	1	0	1	1	11
	1	1	0	0	12
	1	1	0	1	13
	1	1	1	0	14
	1	1	1	1	15

Table 2.5 Variable IDATA1/CDAT1 A/D channel use bit settings

Bit 15	Start Channel	Bit 3	Channel Mode
0	0	0	fixed
1	1	1	alternate

Table 2.6 Variable IDATA1/CDAT1 D/A channel use bit settings

Octal Value	Function
00000K	pulse clock
20000K	DCH clock
40000K	internal clock
60000K	external clock
0-1700K	start channel 0-15
0-17K	final channel 0-15

Table 2.7 Variable IDATA1/CDAT1 octal value bit settings for an A/D operation

Octal Value	Function
04000K	pulse clock
44000K	internal clock
64000K	external clock
10000K	alternate channels
0-1K	start channel 0-1

Table 2.8 Variable IDATA1/CDAT1 octal value bit settings for a D/A operation

The variable IDATA2/CDAT2 also occupies one machine word and specifies the total conversion count. In Fortran, IDATA2 can be an integer or an integer variable. In assembly language, CDAT2 is the label to the word that contains the variable value.

The variable IDATA3/CDAT3 specifies the data array where conversion values are held. In Fortran, IDATA3 is an integer array and it must be placed in a labeled common block when data channel I/O is used. In assembly language, CDAT3 is a label to the storage area that will hold conversion values. The label name must be the same as that given in the SAMGEN created configuration file.

Chapter 3
Configuration Files

Program SAMGEN

The SAMGEN program is used to build a relocatable binary configuration file which is loaded with the A/D/A application program. The configuration file defines to the operating system which hardware and operating modes will be used.

SAMGEN is a Fortran IV program in the form of a save file. This program will run on an Eclipse or Nova computer in either mapped or unmapped environments. It is located in the SAM directory on DP4F. SAMGEN is an interactive program that is executed by name. It takes only a few minutes to complete and is self-explanatory. In addition to producing a configuration .AS file for the assembler, SAMGEN also produces a configuration .SR file which summarizes answers to SAMGEN questions. For most data channel applications, the only parameters needed to run SAMGEN are the device-ids and the name and size of the conversion data buffers.

It is recommended that the configuration files be left in the SAM directory and linked to the user's directory for use. Also, the name given to the configuration file should be of the form, SAMCONFIGxx, where xx is a number unique from other configuration files in the directory.

Sample SAMGEN Dialog

The sample dialog section that follows was executed on

the Eclipse computer. The SAMGEN questions asked would be similar for other configuration files using data channel mode.

The user inputs are noted with the heading "User:" and the system responses are noted with the heading "System:." The user inputs for this dialog are given exactly as stated and the symbol "(CR)" notes that the user should depress the carriage return. At any time during the dialog, SAMGEN can be aborted by typing a CTRL-A followed by a carriage return.

The following commands get into the SAM directory and start up SAMGEN,

System: R

User: DIR DP4F:SAM(CR)

System: R

User: SAMGEN(CR)

SAMGEN will then display an introductory message and ask the following question concerning the configuration file name,

System: Enter a (1-10) character primary file name for this configuration file?

User: SAMCONFIG5(CR)

The configuration file name can be any valid RDOS file name, 10 alphanumeric characters or less, without an extension. SAMGEN will append .AS to the file name for the file to be assembled and .SR to the file name for the file that summarizes SAMGEN dialog. The name should be unique to the current directory or else SAMGEN will abort. The dialog continues as follows,

System: What type OS is your Target system?

Choices are:OS Type:	Response
Mapped RDOS	MRD
Unmapped RDOS	URD
NOVA DOS	URD
micro NOVA DOS	URD
Unmapped RTOS	URT
Mapped RTOS	MRT

Type of OS?

User: MRD(CR)

System: How many DG/DAC chassis (0-4)?

User: 0(CR)

System: Enter the entry name of your SAM fatal error routine (up to 5 characters) or type (CR) for default routine?

User: (CR)

System: What type of computer is your target system?

MN - MicroNOVA

NE - Nova or Eclipse

Enter (MN,NE)?

User: NE(CR)

System: How many Nova/Eclipse Analog Subsystems (# 4330-4333) do you have?

User: 1(CR)

System: What is the device code of A/D converter #1? (5, 15, 21, 23, 31, 35, 41, 45, 51, 55, 61, 65, 71)

User: 21(CR)

The A/D converter is wired for device code 21 octal, which is a Data General standard for the first stand-alone device. The device code for the corresponding D/A converter

is always the A/D code plus two, or 23 in this case. The dialog continues as follows,

System: What mode will you operate the A/D converter in?

AP - specifies Programmed I/O
AD - specifies Data Channel Transfers

Enter (AP,AD):

User: AD(CR)

System: Do you have a special interrupt service routine for this A/D converter?(Y,N):

User: N(CR)

Since the Eclipse is a mapped RDOS system, SAMGEN will now ask questions concerning the name and size of the conversion data buffers. The name given will be the name used in a common block of each application program to align the conversion data array with the data channel map. The size given will be the maximum size that the conversion data array can be dimensioned to in the common block and, hence, specifies the maximum number of conversions a single conversion operation can handle. The data array will be an integer array that receives or sends the conversion values. SAMGEN asks for the A/D data buffer parameters first.

System: Determine the maximum amount of data you expect to transfer in a single request. Then specify the number of 1024 word blocks required to include all of that data.

Enter number of blocks:(1-16)

User: 6(CR)

System: What is the global symbol for your data area? (Enter a 5 character symbol)

User: IBUFF(CR)

Now SAMGEN asks for the D/A data buffer parameters,

```
System:  What mode will you operate the D/A converter
         in?

         BP - specifies Programmed I/O
         BD - specifies Data Channel Transfers

         Enter (BP,BD)?

User:    BD(CR)

System:  Do you have a special interrupt service routine
         for this D/A converter?(Y,N):?

User:    N(CR)

System:  Enter the number of blocks:(1-16):

User:    6(CR)

System:  What is the global symbol for your data area
         (Enter a 5 character symbol):

User:    IBUFO(CR)

System:  Your configuration file is called SAMCONFIG5.SR
         Good luck with your application!  End of SAMGEN
         STOP
         R
```

A printout of the configuration source file for the previous dialog can be obtained by typing the following command line,

```
System:  R

User:    PRINT SAMCONFIG5.SR(CR)
```

The .SR file for the sample dialog is shown in Fig 3-1.

The configuration .AS file can be assembled in the SAM directory. The directions given in Chapter 6 for assembling application programs can be applied to assembling the configuration file. Assembling the configuration file produces a relocatable binary file which must be loaded with the application program in the RLDR command line. A link, such as the following for the sample file, should be set up in the directory that the user will be working in. Application programs can then be loaded from that directory,

```
SAMCONFIG5.RB          SAM:SAMCONFIG5.RB
```

```
;      SAMGEN  Rev.   2.10  11/27/82 at 19: 8    Filename: SAMCONFIG5.SR

;Answers you gave in the SAMGEN dialog are shown in comment lines.
;Your inputs are immediately preceded by a colon (:) and appear
;in the same order as you gave them to SAMGEN.

; Target operating system type :MRD
; Number of DG/DAC 4300 chassis  configured: 0
; Fatal error handler name   : -1
; Fatal error handler mailbox: -1

        DCB.X    SAMCO  100 -1    -1


; Number of Analog Subsystem  :1



; A/D Con. #1  Device Code :21  Mode :AD   Fortran ID =  IDS21

; External interrupt handler specified :(NONE)
; Number of pages in Data Channel area : 6
; Specifying a starting address for Data Channel area :Y
; Data Channel starting address :IBUFF

        DCB.M  DBS21  D.IDF+D.INF+D.DCH   21
        DCB.I  DTS21  SAINI      6.  IBUFF
        DCB?C    -1    -1   DSS21
        DCT.M  DTS21  000377  INTSA  DSS21

        DCB.N    S21  D.FIF     21     00     AD
        DCB.S  DBS21    0     AD.IS  AD.IN  SAIRT
        DCB.A



; D/A Con. #1  Device Code :23  Mode :BD   Fortran ID =  IDS23

; External interrupt handler specified :(NONE)
; Number of pages in Data Channel area : 6
; Specifying a starting address for Data Channel area :Y
; Data Channel starting address :IBUF0

        DCB.M  DBS23  D.IDF+D.INF+D.DCH   23
        DCB.I  DTS23  SAINI      6.  IBUF0
        DCB?C    -1    -1   DSS23
        DCT.M  DTS23  000377  INTSA  DSS23

        DCB.N    S23  D.FIF     23     00     BD
        DCB.S  DBS23    0     BD.IS  BD.IN  SAIRT
        DCB.A

        DCB.E

;End of SAMGEN configuration file.
```

Fig 3-1 Sample dialog .SR file

Chapter 4
Fortran V Operation

This chapter describes the source code necessary to operate the Eclipse A/D/A device in Fortran V application programs. It is divided into three sections according to the purpose of the source code--setup, initialization, and conversion. The SAM error codes for the Fortran error return variables are given in Table 4.1. A sample program that could be used with the configuration file built in Chapter 3 is given in Fig 4-1.

Setup

The source code for a typical setup in data channel mode is shown below,

```
EXTERNAL IDS21
EXTERNAL IDS23
COMMON / IBUFF / IDATA3(16384)
COMMON / IBUFO / IDATO(1024)
DIMENSION IORBA(16)
```

The application program must declare both device-ids, even if the application program will only use one device. Accordingly, the common block for both the A/D and D/A data buffers must be declared, even if only one type of conversion will be performed. The data arrays in the common blocks can be dimensioned less than the number given to SAMGEN when creating the configuration file. They must not, however, be dimensioned larger than the number given to SAMGEN. The space for the data buffers can be divided among more than one data array as long as the cumulative space does not exceed the number

given to SAMGEN. A single conversion operation, however, can only operate on one data array. The IORBA data array can be any integer data array dimensioned to at least 16. It is used by SAM as a scratchpad for processing the conversion operation that references it. SAM also uses certain elements of this array to convey information concerning the status of the conversion operation. This will be discussed in the conversion operation section.

Initialization

Prior to issuing any conversion call, the application program must issue the DSTRT call shown below,

```
CALL DSTRT(IER)
```

The variable IER is the standard Fortran error return variable. This call initializes the operating system for conversion operations. It can be located anywhere in the application program before the first conversion call, but must be given only once in the same application program.

Conversion

The DOIT[/W] call is used to request a conversion operation. It has the basic form shown below,

```
CALL DOIT[/W] (IORBA,device-id,8,IDATA1,IDATA2,IDATA3,IER)
```

The call has two forms--DOIT or DOITW. The DOITW call is used to halt program control until the operation requested is completed. The DOIT call is used to instigate the operation and returns program control immediately. With this call, the DREC[/W] call must be used later in the program to determine

if the conversion operation has completed. The device-id is IDS21 for an A/D operation and IDS23 for a D/A operation. The variables IDATA1, IDATA2, and IDATA3 must be specified as given in Chapter 2. The variable IER is the standard Fortran error return variable. The value of IORBA(14) should always be checked after completion of a DOIT[/W] call in data channel mode to determine if an external interrupt or if a clock overrun/underrun occurred. The IER variable will not return an error for either of these two conditions. On a normal return, all the bits of IORBA(14) should be zero except bit 1. This will give a value of 40000K to IORBA(14). If an external clock interrupt occurred which caused processing to be aborted before the conversion operation was completed, then bit 0 of IORBA(14) will be set to one. If a clock overrun occurred on an A/D operation or a clock underrun occurred on a D/A operation, then bit 8 of IORBA(14) will be set to one. If IORBA(14) returns a value that does not indicate a normal return or either of these two error conditions, the conversion data should be checked if possible to determine if it is plausible. The SAM User's Manual (p. 4-29,34) does not cover "non-normal" returns other than the two given.

The DREC[/W] call is used to check for completion of a DOIT conversion operation. It has the basic form shown below,

CALL DREC[/W] (IORBA,IER)

The call has two forms--DREC or DRECW. The DRECW call is used to halt program control until the DOIT conversion operation, with the IORBA array referenced, is completed. The

DREC call is used only to check the current status of the conversion operation. If the operation is completed, then the DREC call places a non-zero value in IORBA(6). If the operation has not completed, then the DREC places a value of zero in IORBA(6). A guide illustrating the different DOIT[/W] and DREC[/W] combinations is given in Fig 4-2.

Value	Meaning
2179	No -.LNK routine in DCB, invalid DCB. Often results from an invalid device-id, so check the device-ids. The first two characters are ID, the third either S, A, or O, and the last two are numbers (e.g., IDS21).
2180	No DCB identifier in IORB, invalid DCB. Same cause as 2179.
2181	Not used. This error should not occur.
2184	No initializing routine for a device that needs initialization. Same cause as 2179.
2185	Output requested to a channel for an illegal device (e.g., output to an A/D converter).
2186	Attempt to set up a locked IORB array. This can happen if a second DSAN/DSOR call uses the same IORB array argument before the original DSAN/DSOR completes.
2187	Unable to find free IORB block in IORB array. Can happen if the IORB array was DIMENSIONed too small. A multiple-operation call needs 8 elements + 8 elements per operation.
2188	No DCB exists with specified device-id. Same cause as 2179.
2189	Attempt to use unsupported feature (e.g., mapped call in unmapped system).
2190	Attempt to return bad buffer. Will never occur.
2191	An IDATAx argument gave an illegal clock setting for an A/D or D/A converter.

Table 4.1 SAM Fortran error codes
(SAM User's Manual, p. 6-9)

Value	Meaning
2192	Illegal conversion count -- more than 255 or less than 1 -- for an A/D converter mode in A2; DG/DAC only.
2193	Assembly language only. Attempt to move data channel map while IORB is locked. A task tried to change the map while a request was using the window.
2194	Attempt to move data channel map to an address outside the window.
2195	Illegal conversion count: less than 1 or more than the device allows.
2196	Interrupt occurred from 4222 without a strobe or latch change.
2197	Assembly language only. Attempt to use data channel map while it is being initialized or moved.
2198	Assembly language only. Data channel not initialized; use an RMAP call before issuing this mode A2 request.
2199	SAM panic code. SAM could not transmit (.IXMT) to the calling task on IORB array completion. SAM aborts the program unless you set up a fatal error handling RECeive task and gave its name to SAMGEN, as described in Chapter 5, "Initial Dialog".
2200	External interrupt occurred on a stand-alone analog converter, aborting the request. This error returns from ISA calls only, not from DSAN/DSOR calls.

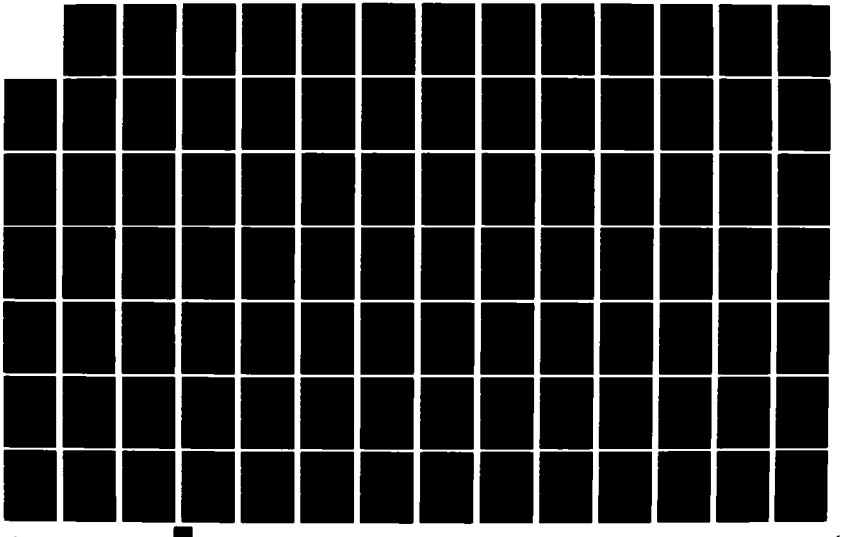
Table 4.1 continue

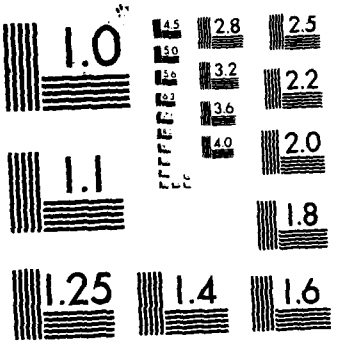
```
C     This program will collect and then output 5120 data samples
C     using the Eclipse A/D/A device.

      EXTERNAL IDS21    ;setup code
      EXTERNAL IDS23
      COMMON / IBUFF / IDATA3(5120)
      COMMON / IBUFO / IDATO(5120)
      DIMENSION IORBA(16)

      CALL DSTRT(IER)   ;always initialize device
      IF (IER.NE.1) CALL ERROR("DSTRT")

      IDATA1=60000K     ;external clock and channel 0 (A/D)
      IDATA2=5120

      TYPE "Press carriage return to begin A/D"
      ACCEPT

      CALL DOITW(IORBA,IDS21,8,IDATA1,IDATA2,IDATA3) ;A/D

      IF (IER.NE.1) TYPE "DOIT error ",IER
      IF (IORBA(14).NE.40000K) WRITE(10,1) IORBA(14)
1     FORMAT("IORBA(14): ",O6)

      DO 25 I=1,5120    ;load sampled data into D/A array
      IDATO(I)=IDATA3(I)
25    CONTINUE

      IDATA1=64000K     ;external clock and channel 0 (D/A)

      TYPE "Press carriage return to begin D/A"
      ACCEPT

      CALL DOITW(IORBA,IDS23,8,IDATA1,IDATA2,IDATO,IER) ;D/A

      IF (IER.NE.1) TYPE "DOIT error ",IER
      IF (IORBA(14).NE.40000K) WRITE(10,1) IORBA(14)

      CALL EXIT
      END
```

Fig 4-1 Sample Fortran A/D/A program

AD-A124 750 EXPANSION OF THE ECLIPSE DIGITAL SIGNAL PROCESSING SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING G R ALLEN DEC 82 2/3

UNCLASSIFIED AFIT/GE/EE/82D-16 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

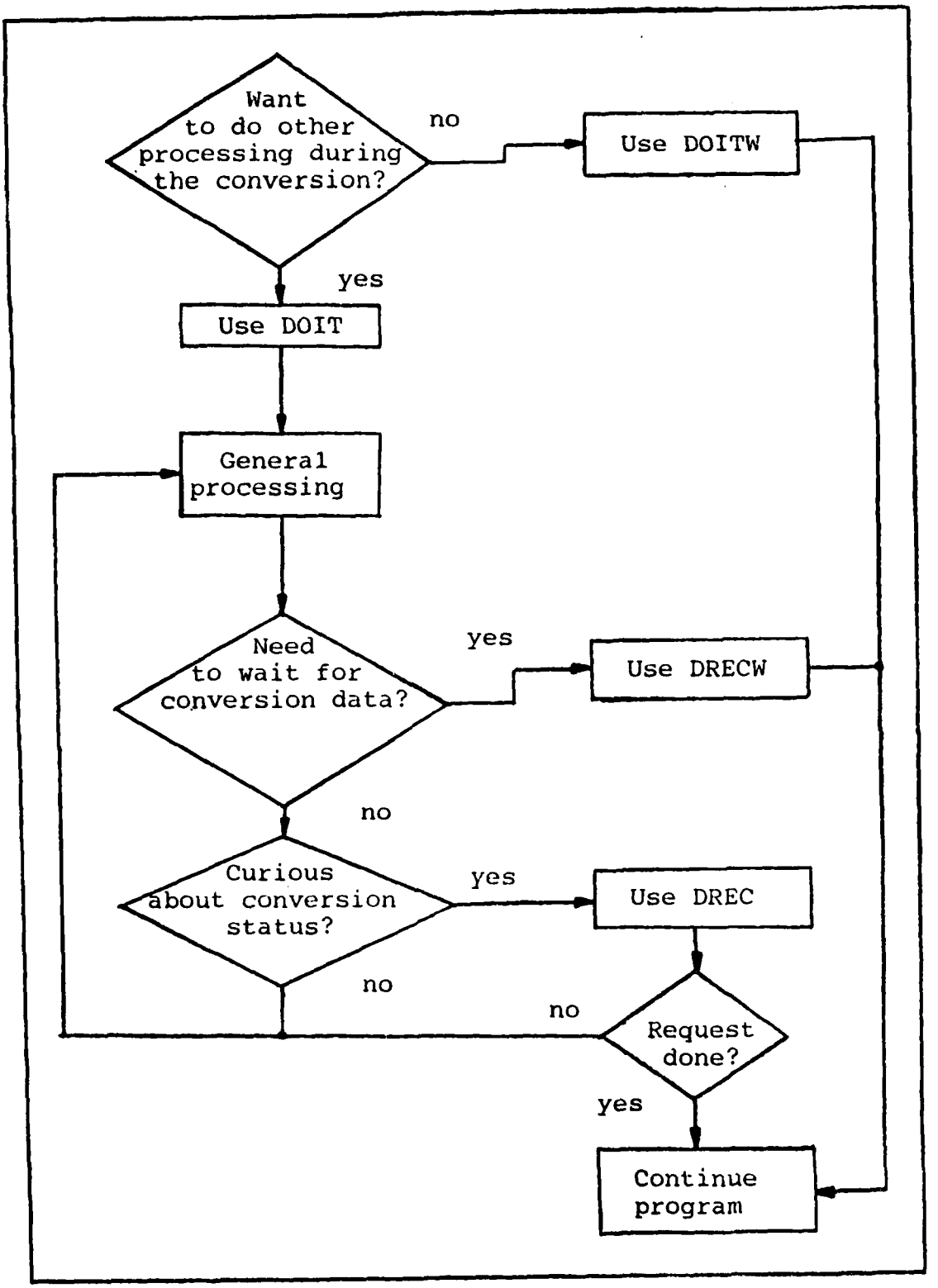


Fig 4-2 DOIT[/W] and DREC[/W] options

Chapter 5
Assembly Language Operation

This chapter describes the source code necessary to operate the Eclipse A/D/A device in Data General assembly language. It is divided into three sections according to the purpose of the source code--setup, initialization, and conversion. The SAM libraries provide various macros that make operating the device at the assembly language level much easier. A macro is a predefined section of code and in this chapter it is used in a manner similar to a Fortran subroutine. The Models 4330-4333 Programmer's Reference should be consulted for operating the device in assembly language without the macros described in this chapter. The SAM error codes for assembly language error return messages are given in Table 5.1. A sample program that could be used with the configuration file built in Chapter 3 is given as Appendix A. Throughout this chapter, references will be made to SAM Fortran subroutines. This is done for further clarification, since it is assumed that most users will operate the device at the Fortran level first. This, of course, is not required.

Setup

Shown on the next page is the source code for a typical setup in data channel mode.

```
        .ENT    IBUFF           ;declare the A/D data buffer
        .ENT    IBUFO           ;declare the D/A data buffer
        .ZREL
RECW    0
        .NREL
IORB    .BLK    10
STACK   .BLK    40
IBUFF   .BLK    12000           ;set aside 6KW for A/D storage
IBUFO   .BLK    12000           ;set aside 6KW for D/A storage
```

The application program must declare both the A/D and D/A data buffer names given to the SAMGEN program with the .ENT statement. This allows the operating system to access the conversion storage area in the application program. Accordingly, both data buffer names must appear in the program and label the areas that will be used for conversion data storage. The .BLK statement can be used to set aside the storage area up to the limit specified in the SAMGEN created configuration file. The IORB variable is an eight-word block that holds the parameters for a conversion operation. The STACK variable is a 32-word block that is used by SAM as a scratchpad to process a conversion operation. The RECW variable is used to indicate when a conversion operation is completed. Each separate single-operation conversion request must have its own IORB, STACK, and RECW setup.

Initialization

Prior to issuing any conversion call, the application program must issue the S.STR macro shown below,

```
S.STR   STACK
JMP     ERR1                    ;to handle error return
JMP     CONT1                   ;to continue processing
```

This call initializes the operating system to the device and is similar in function to the Fortran DSTRT call. Program

control is returned to the location immediately following the macro on an error return. The error code is contained in accumulator 2. For a normal return, program control is returned to the location immediately following the macro plus one.

Conversion

Performing a conversion operation in assembly language that is similar to the Fortran DOIT call, is a two-step process. First, the parameters of the conversion call are placed in the IORB block with the S.SET macro. Then, either the S.DOR or S.DAN macro can be used to initiate the single-operation request. The S.DAN macro has been chosen to be used for the purpose of illustration. The basic form of the S.SET macro is shown below,

```
S.SET   IORB,0,device-id,CDAT1,CDAT2,CDAT3
```

The device-id is IDS21 for an A/D operation and IDS23 for a D/A operation. The variables CDAT1, CDAT2, and CDAT3 must be specified as given in Chapter 2. The basic form of the S.DAN macro to initiate the conversion operation is shown below,

```
S.DAN   IORB,RECW,STACK
JMP     ERR2                ;to handle error return
JMP     CONT2               ;to continue processing
```

As with the S.STR macro, program control is returned to the location immediately following the S.DAN macro on an error return. The error code is contained in accumulator 2. For a normal return, program control returns to the location immediately following the macro plus one. Completion of the conversion operation is determined by checking the status

of the variable RECW. When the operation is completed, SAM places a non-zero value in RECW.

As is the case with Fortran operation, the error return will not indicate if an external interrupt or a clock overrun/underrun occurred. Either of these conditions may make the conversion data invalid. The CDAT1 word can be checked to determine if either of these conditions occurred. On a normal return, all of the bits of CDAT1 are zero except bit 1. If an external interrupt occurred which caused processing to be aborted before the conversion operation completed, then bit 0 of CDAT1 will be set to one. If a clock overrun occurred on an A/D operation or a clock underrun occurred on a D/A operation, then bit 8 of CDAT1 will be set to one.

Program control can be held up until the conversion operation is completed, as is done with the Fortran DOITW call, by using the .REC macro. The basic setup for the .REC macro is shown below,

```
.EXTN     .REC
.ZREL
PREC      RECW
.NREL
LDA       0,PREC
.REC
```

The address of the RECW word is placed in accumulator 0 and the macro .REC is called. Program control is then held at the location of .REC until the conversion operation referenced with the PREC word is completed. The value of the word at label RECW should be 0 prior to the .REC call.

Value = mnemonic	Meaning
4200 = DER00	No -.LNK routine in DCB, invalid DCB. Could be an invalid (mistyped) SAMGEN device-id. All device-ids begin with the letters ID, followed by S, A, or O, followed by two numbers.
4201 = DER01	No DCB identifier in IORB, invalid DCB. Same cause as 4200.
4202 = DER02	Not used. This should not occur.
4205 = DER05	No initialization routine for a device that needs initialization. Same cause as 4200.
4206 = DER06	Output requested to a channel for an illegal device (e.g., output to an A/D converter).
4207 = DER07	FORTRAN only. Attempt to set up a locked IORB array.
4210 = DER10	FORTRAN only. Unable to find free IORB in IORB array.
4211 = DER11	No DCB exists with specified device-id. Same cause as 2179.
4212 = DER12	Attempt to use unsupported feature (e.g., mapped call in unmapped system).
4213 = DER13	Attempt to return bad buffer. Will never occur.
4214 = DER14	Illegal clock setting for an A/D or D/A converter.

Table 5.1 SAM assembly language error codes (SAM User's Manual, p. 9-4)

Value = mnemonic	Meaning
4215 = DER15	Illegal conversion count -- more than 377_8 or less than 1 for an A/D converter mode A2; DG/DAC only.
4216 = DER16	Attempt to move data channel map while IORB is locked. A task tried to change the map while a request was using the window.
4217 = DER17	Attempt to move data channel map to an address outside the window.
4220 = DER20	Illegal conversion count: less than 1 or more than the device allows.
4221 = DER21	Interrupt occurred from 4222 without a strobe or latch change.
4222 = DER22	Attempt to use data channel map while it is being initialized or moved.
4223 = DER23	Data channel not initialized; use an RMAP call before issuing this mode A2/AD request.
4224 = DER24	SAM panic code. SAM could not transmit (.IXMT) to the calling task on IORB array completion. SAM aborts the program unless you set up a fatal error handling RECeive task and gave its name to SAMGEN, as described in Chapter 5, "Initial Dialog".
4225 = DER25	ISA calls only. External interrupt occurred on a stand-alone analog converter, aborting request.

Table 5.1 continue

Chapter 6
Running Application Programs

This chapter describes how to compile and load SAM application programs on the Eclipse computer. The RDOS commands given in this chapter should only be typed when the system R prompt is given on the screen. The "(CR)" denotes that the user should depress the carriage return and "program" denotes the user's program name.

Compiling

Compiling a SAM Fortran V application program is identical to compiling the usual Fortran V program. The directory must contain the necessary files or links to them for Fortran V compiling. The following command line will compile the application program,

FORTRAN program

The following files are required to assemble a SAM assembly language application program,

MAC.SV,NBID.SR,OSID.SR,NEID.SR,ARDOS.SR,PARU.SR
and SAMPARS.SR

All of the above files are or should be in the SAM directory. Links can be made to any of these files that are not contained in the directory that the user will be working in. The command line given below will assemble the application program.

```
MAC NBID/S OSID/S NEID/S ARDOS/S PARU/S↑(CR)
 SAMPARS/S program(CR)
```

The symbol "↑" is used in the previous command line, since it is the correct syntax for continuing an RDOS command on

the next line. The command, however, can be typed on a single line if it fits.

To lessen the effort in assembling an application program, an indirect file named SAMASSM has been created and placed in the SAM directory. It contains the parameter files as typed in the previous command line. Using this indirect file, the previous command line can be given as shown below,

```
MAC @SAMASSM@ program
```

As is the case with all indirect files, SAMASSM can be linked to the directory that the user will be working in. However, links to the files contained in an indirect file must still be made.

Loading

The files given below are required to load the application program.

```
samconfig,SAMF5E.LB,SAME.LB,TFLIB
```

The file "samconfig" denotes the user's relocatable binary configuration filename. The other files are or should be in the SAM directory. Links can be made to any of these files that are not contained in the directory that the user will be working in. The command line given below will load the application program.

```
RLDR/P 2/K program config samconfig subroutines ↑(CR)
 SAMF5E.LB SAME.LB @TFLIB@
```

In the previous command line, "subroutines" denotes where any user subroutines should be loaded. The P switch is

optional. If it is attached, the load addresses for the various modules in the program are given in octal format. The 2/K switch is required and creates a second task. The SAM package is designed for a multitask environment and requires at least one more task than the program uses.

To lessen the effort in loading an application program, an indirect file named SAMLIB has been created and placed in the SAM directory. It contains the library and TFLIB files as typed in the above command line. Using this indirect file, the previous command line can be given as shown below,

RLDR/P 2/K program samconfig subroutines @SAMLIB@

Every application program will generate an "XN I43RT" error on the load command. This is because the samconfig file is requesting software to handle an A/D/A device other than the model 4331. Since the device that the samconfig file is requesting information for is not in the system and will not be called upon, this error message can be ignored. This is the only error message from the load command that can be ignored.

```
;       This program will collect and then output 5120 data
;       samples using the Eclipse A/D/A device. It is
;       similiar in operation to the sample Fortran A/D/A
;       program, except that the user cannot initiate
;       both conversion operations. The user can only
;       initiate the A/D operation by turning the external
;       clock on when it is desired to begin collecting
;       data. The CDAT1 variable for both, the A/D
;       and D/A operation, has been set for external
;       clock and channel 0.

        .ENT    IBUFF
        .ENT    IBUF0
        .EXTN   .REC
        .ZREL
PBEG:   BEG     ;addresses
PBYE:   BYE
PADO:   ADO
PDAO:   DAO
PER1:   ER1
PER2:   ER2
PER3:   ER3
PIBUF:  IBUFF
PIBU0:  IBUF0
PREC:   RECW
RECW:   0
CDAT2:  12000   ;conversion count
ZERO:   0
.ER:    ERROR   ;jump locations
.BNOC:  BNOC
.C"`R:  CHAR
.NWLN:  NWLN
.MSSG:  MSSG
        .NREL

;       Main program

START:  LDA     0,PBEG  ;send startup message to screen
        JSR     @.MSSG
        JSR     @.NWLN

        S.STR   STACK   ;initialize the device
        JMP     ERR1    ;error return will have error code in AC2.

        S.SET   IORB,0,IDS21,60000,CDAT2,IBUFF ;setup the A/D

        LDA      0,PADO ;send A/D message to screen
        JSR      @.MSSG
        JSR      @.NWLN

        S.DAN   IORB,RECW,STACK ;initiate the A/D
        JMP     ERR2    ;error return will have error code in AC2.
```

```
        LDA     0,PREC  ;wait for the A/D to
        .REC            ;complete.

        JMP     .+4     ;load the sampled data into
ADDR1:  0               ;the D/A data array
ADDR2:  0
COUN:   0
        LDA     0,PIBUF
        STA     0,ADDR1
        LDA     0,PIBUO
        STA     0,ADDR2
        LDA     3,CDAT2
        STA     3,COUN
        LDA     3,@ADDR1
        STA     3,@ADDR2
        ISZ     ADDR1
        ISZ     ADDR2
        DSZ     COUN
        JMP     .-5

        LDA     0,ZERO  ;zero the status word
        STA     0,RECW

        S.SET   IORB,0,IDS23,64000,CDAT2,IBUFO ;setup the D/A

        LDA     0,PDAO  ;send D/A message to screen
        JSR     @.MSSG
        JSR     @.NWLN

        S.DAN   IORB,RECW,STACK ;initiate the D/A
        JMP     ERR3

        LDA     0,PREC  ;wait for the D/A to
        .REC    ;complete

        LDA     0,PBYE  ;send exit message to screen
        JSR     @.MSSG
        JMP     DONE

ERR1:   LDA     0,PER1  ;send error message to screen.
        JMP     NUMB

ERR2:   LDA     0,PER2
        JMP     NUMB

ERR3:   LDA     0,PER3
NUMB:   JSR     @.MSSG
        MOV     2,0     ;get error code from AC2
        JSR     @.BNOC

DONE:   JSR     NWLN
        .SYSTM
        .RTN
        JMP     @.ER
```

```
;       The NWLN routine places the cursor at the beginning of
;       the next line.

NWLN:   JMP     .+4
STOR1:  0
CR:     15
NL:     12
        STA     3,STOR1
        LDA     0,CR
        JSR     @.CHAR
        LDA     0,NL
        JSR     @.CHAR
        JMP     @STOR1

;       The CHAR routine sends an ASCII character that
;       has been placed in AC0 to the screen. The
;       contents of AC3 are destroyed.

CHAR:   JMP     .+2
STOR2:  0
        STA     3,STOR2
        .SYSTM
        .PCHAR
        JMP     @.ER
        JMP     @STOR2

;       The MSSG routine sends a message that has been
;       created with the .TXT pseudo-op to the screen.
;       The label for the message must be placed in
;       AC0.

MSSG:   JMP     .+4
STOR3:  0
STOR4:  0
WORD1:  177
        STA     3,STOR3
        STA     0,STOR4
        LDA     0,@STOR4
        LDA     3,WORD1
        AND     0,3,SNR
        JMP     .+12
        JSR     @.CHAR
        LDA     0,@STOR4
        MOVS    0,0
        LDA     3,WORD1
        AND     0,3,SNR
        JMP     .+4
        JSR     @.CHAR
        ISZ     STOR4
        JMP     MSSG+6
        JMP     @STOR3

;       The BNOC routine will convert a 16-bit binary integer
```

```
;       placed in AC0 to ASCII character string for output.
;       The ASCII string is output in reversed order.

BNOC:   JMP     .+7
STOR5:  0
STOR6:  0
SIX:    6
COUNT:  0
WORD2:  7
WORD3:  60
        STA     3,STOR5
        STA     0,STOR6
        LDA     0,SIX
        STA     0,COUN
        LDA     0,WORD2
        LDA     3,STOR6
        AND     3,0
        LDA     3,WORD3
        ADD     3,0
        JSR     @.CHAR
        LDA     0,STOR6
        MOVZR   0,0
        MOVZR   0,0
        MOVZR   0,0
        STA     0,STOR6
        DSZ     COUN
        JMP     BNOC+13
        JMP     @STOR5

;       Storage space

ER1:    .TXT    *S.STR ERROR : *
ER2:    .TXT    *S.DOR ERROR (A/D): *
ER3:    .TXT    *S.DOR error (D/A):*
BEG:    .TXT    *Hello*
BYE:    .TXT    *Good Bye*
ADO:    .TXT    * -- A/D --*
DAO:    .TXT    * -- D/A --*

IORB:   .BLK    10
STACK:  .BLK    40
IBUFF:  .BLK    12000
IBUFO:  .BLK    12000

;       Error return

ERROR:  .SYSTM
        .ERTN
        JMP     @.ER

        .END    START
```

Appendix B

Extended Memory Data Collection Measurements

Eclipse A/D/A Device Extended Memory Data Collection Measurements

Background

There is a limit on the number of data samples that the Eclipse A/D/A device can collect on a single A/D call. This limit is due to the size of the A/D data buffer. Samples are lost during the time it takes to remove data from this buffer and issue another conversion call. The fastest method of moving data on the Eclipse is with the extended memory feature. With this feature data is not physically moved, address registers are simply changed. Program SPEECH will be used to collect data in long mode operation and the remap intervals will be noted.

Purpose

The purpose of this test is to determine the number of sample points lost between remap operations. The affect of activity on the opposite ground will also be noted.

Pretest

Since a linear test signal will allow the delta voltage between samples to be easily seen, a triangle wave will be used as the test signal.

To best illustrate the time lost during a remap operation, the sampling rate will be set near maximum. From the Technical Reference of Analog Data Subsystems, Models 4330-4333, p. 7, the maximum A/D conversion rate is given as 22KHz (45.4 microsec). The test signal will be sampled at 21KHz (47.6

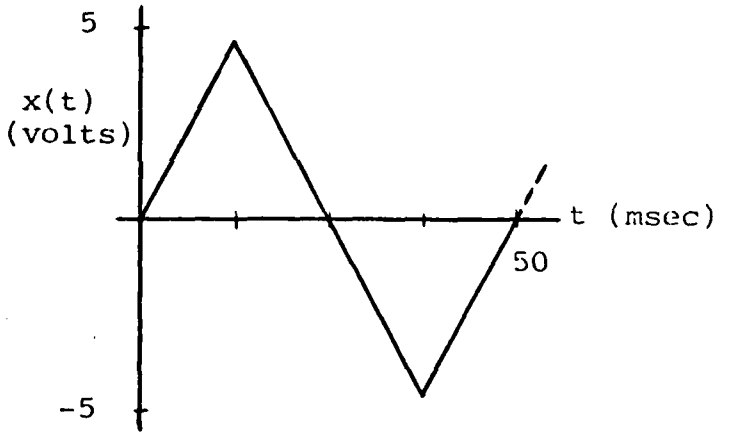
microsec).

A plot routine will be used that can plot a maximum of 512 points. The signal frequency will be set at a rate which will collect 512 samples on a single peak-to-peak swing, that is, one-half of a period. The test signal period to allow this can be calculated as follows,

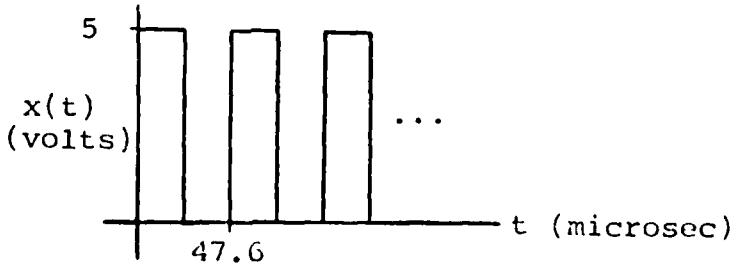
47.6 microsec/pts * 1024 pts = 48.76 msec (20.51Hz)

The test signal frequency will be set to 20Hz, which will closely satisfy the above condition and certainly satisfy the Nyquist sampling condition.

The test signal and clock signal to be used are shown below,



test signal



clock signal

An oscilloscope will be used to view both signals with the settings as below,

	distance		scale	value
clock:	4.76 cm	*	10 microsec/cm	= 47.6 microsec (21KHz)
signal:	5.0 cm	*	10 msec/cm	= 50.0 msec (20Hz)

A frequency counter will be used to verify the above frequencies.

Test Equipment

	PME I.D. Number	Date of Cal.	Date of Recal.
Oscilloscope, Ballantine	49100/ 4H5627	27 Jul 82	14 Nov 82
Frequency Counter, HP 5326A Counter-Timer	49100/ 4H6122	28 Jul 82	25 Nov 82
External Clock, Wavetek Generator	49100/ 4H6222	22 Mar 82	22 Mar 83
Test Signal, Wavetek Generator	49100/ 4H6008	19 Nov 81	19 Nov 82

Test Comments and Results

1. Test equipment settings were set as described in Pretest and frequency settings were verified with a frequency counter.
2. Program SPEECH was executed on the background in long mode operation and three data files were collected under the following conditions,

DATA1: foreground inactive (CTRL-F)
DATA2: foreground active and idle
DATA3: foreground active and compiling a program

3. The three data files were changed from two's complement data to real number data with program CNVRT. The new data files were named as follows,

   TESTDATA1: DATA1 converted
   TESTDATA2: DATA2 converted
   TESTDATA3: DATA3 converted

4. Program PLOT was used to view the remap regions in the TESTDATAx files.

5. The frequency of the external clock generator was slightly readjusted between test runs one and two.

Conducted by: Lt Allen

Date: 11 Oct 82

Appendix C

Source Code
for
A/D/A Operations Software

```
C*******************************************************************************
C     Title: Speech
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This is the central program of a six-program package that utilizes
C        the Eclipse A/D/A device to work with speech data files. This is
C        an interactive software package that allows the user to collect,
C        edit, and play back speech data files.

C     Environment:
C        This is a Fortran V program that has been desigined to run
C        on a mapped-RDOS Eclipse S/250 minicomputer equipped with a
C        model 4331 single board converter.

C     Compile command:
C        FORTRAN SPEECH

C     Load command:
C        RLDR/P SPEECH NEWSCR @FLIB@

C     Comments:
C        Refer to lines 30 and 47 of the program text for information
C        explaining the operation of this package.
C        The term "block" is used throughout the program text to refer
C        to a disk block (a disk block contains 256 bytes).

C*******************************************************************************

      INTEGER OPTION,MENU2,MENU3,MENU4,MENU5
      INTEGER WORK(15872)
      INTEGER EDBUF,DTBUF,EMPTY,FULL
      INTEGER FILENAM(7),STATUS(18)
      INTEGER NUMBLK,FIRBLK,STBLK,BLKLEFT,READBLK,DBLKS,FBLKS,WBLKS
      INTEGER MODE,SHORT,LONG

      DATA EDBUF,DTBUF,EMPTY,FULL,SHORT,LONG / 3*0,2*1,2 /
      DATA WORK / 15872*0 /

      CALL NEWSCR    ;erase the screen
C
C     Erase files that may have been left from a previous abort.
C
      CALL DFILW("DIGI.DT",IER)
      CALL DFILW("DIGI.ED",IER)
      CALL DFILW("DIGI.OU",IER)
      CALL CFILW("DIGI.DT",2,IER)
      CALL CFILW("DIGI.ED",2,IER)
      CALL CFILW("DIGI.OU",2,IER)

 30   TYPE "(CR)
```

```
     *If this program fails to operate properly, consult the(CR)
     *laboratory software documentation manual for a complete(CR)
     *description of hardware setup and verify all connections.(CR)
     *(CR)
     *This program maintains two independent buffers, called data(CR)
     *and edit, to hold conversion data.(CR)
     *(CR)
     *Conversion data can be placed in the data buffer through an(CR)
     *A/D operation or by a series of read operations from disk(CR)
     *files. The data buffer can then be copied into the edit buffer(CR)
     *for editing. Editing operations do not affect the data buffer.(CR)
     *(CR)
     *To select menu options in this program, type only the number(CR)
     *corresponding to the option and carriage return.(CR)
     *(CR)
     *Press carriage return to begin."
      ACCEPT
      CALL NEWSCR
C
C     The variable MODE is used to determine which program wil be called
C     to input or output the data buffer.
C
 45   ACCEPT "(CR)
     *Please select which mode will be used or a description of(CR)
     *both modes,(CR)
     *   1: short mode(CR)
     *   2: long mode(CR)
     *   3: description(CR)
     *selection:",MODE
      CALL NEWSCR
      IF (MODE.EQ.SHORT .OR. MODE.EQ.LONG) GO TO 49
      IF (MODE.EQ.3) GO TO 47
      WRITE(10,1)
      GO TO 45
C
C     This message can be requested by the user to explain the difference
C     between short mode and long mode operation.
C
 47   TYPE "(CR)
     *Acording to the length of the data file, this program(CR)
     *operates in one of two modes- short mode or long mode.(CR)
     *(CR)
     *Short mode operation allows 15,872 samples (1.98 sec at 8 KHz(CR)
     *sampling) to be collected and played back without interruption.(CR)
     *The directory should have 175 disk blocks available for(CR)
     *execution in this mode.(CR)
     *(CR)
     *Long mode operation allows 51,200 samples (6.40 sec at 8 KHz(CR)
     *sampling) to be collected and played back, but with short(CR)
     *interrupts every 10,240 samples. For minimum interrupt time,(CR)
     *it should be operated on the background terminal with the(CR)
     *foreground terminal brought down. Long mode operation requires(CR)
     *40 KW of extended memory and has longer processing delays(CR)
     *than short mode. The directory should have 450 disk blocks(CR)
```

```
     *available for execution in this mode.(CR)
     *(CR)
     *Press carriage return to continue."
      ACCEPT
      CALL NEWSCR
      GO TO 45
C
C     The variable MODE is communicated to the editor program through
C     file DIGI.OU. This allows the editor program to determine which
C     program to call to output the edit buffer.
C
 49   CALL FOPEN(1,"DIGI.OU")
      CALL WRSEQ(1,MODE,2,IER)
      CALL FCLOSE(1)

C*******************************************************************************

C
C     This is the central program's main menu.
C
 50   ACCEPT "(CR)
     *Please select which operation will be performed,(CR)
     *   1: A/D conversion into data buffer(CR)
     *   2: D/A conversion out of data buffer(CR)
     *   3: editing(CR)
     *   4: read from file to data buffer(CR)
     *   5: write data buffer to file(CR)
     *   6: copy data buffer to edit buffer(CR)
     *   7: exit(CR)
     *selection:",OPTION

      CALL NEWSCR
      IF (OPTION.EQ.1) GO TO 100
      IF (OPTION.EQ.2) GO TO 200
      IF (OPTION.EQ.3) GO TO 205
      IF (OPTION.EQ.4) GO TO 400
      IF (OPTION.EQ.5) GO TO 700
      IF (OPTION.EQ.6) GO TO 600
      IF (OPTION.EQ.7) GO TO 1000
      WRITE(10,1)
 1    FORMAT("(CR)(7)(CR)(7)(CR)(7)
     *Please select options from the list only.")
      GO TO 50

C*******************************************************************************

C
C     This section of code fills the data buffer with an A/D conversion
C     operation.
C
 100  DTBUF=FULL   ;this operation will put data in the data buffer
      TYPE "(CR)
     *   --) entering A/D mode (--"
      IF (MODE.EQ.SHORT) CALL SWAP("SMALLIN.SV",IER)
```

```
      IF (MODE.EQ.LONG) CALL SWAP("BIGIN.SV",IER)
      IF (IER.NE.1) TYPE "SWAP error ",IER," with A/D file"
      GO TO 50

C*************************************************************************

C
C     This section of code outputs the data buffer with a D/A conversion
C     operation.
C
 200  IF (DTBUF.EQ.EMPTY) GO TO 251
      TYPE "(CR)
     *   --) entering D/A mode (--"

      CALL DFILW("DIGI.FG",IER)
      IF (MODE.EQ.SHORT) CALL SWAP("SMALLOUT.SV",IER)
      IF (MODE.EQ.LONG) CALL SWAP("BIGOUT.SV",IER)
      IF (IER.NE.1) TYPE "SWAP error ",IER," with D/A file"
      GO TO 50

C*************************************************************************

C
C     This section of code calls the editor program.
C
 205  IF (EDBUF.EQ.EMPTY) GO TO 250
      TYPE "(CR)
     *   --) entering edit mode (--"
      CALL SWAP("EDITOR.SV",IER)
      IF (IER.NE.1) TYPE "SWAP error ",IER," with editor file"
      GO TO 50

C*************************************************************************

C
C     One of the following messages is sent to the screen if the user
C     attempts a buffer operation when the buffer is empty.
C
 250  TYPE "(CR)(7)(7)(7)
     *The edit buffer is currently empty."
      GO TO 50

 251  TYPE "(CR)(7)(7)(7)
     *The data buffer is currently empty."
      GO TO 50

C*************************************************************************

C
C     This section of code allows the user to fill the data buffer through
C     a series of read operations from disk files.
C
 400  ACCEPT "(CR)(7)(7)(7)
     *The current data buffer is erased(CR)
```

```
     *prior to reading from disk.(CR)
     *(CR)
     *Do you want to,(CR)
     *   1: continue(CR)
     *   2: return to the main menu(CR)
     *selection:",MENU5

      IF (MENU5.EQ.1) GO TO 401
      IF (MENU5.EQ.2) GO TO 50
      WRITE(10,1)
      GO TO 400

 401  IF (MODE.EQ.SHORT) TYPE "(CR)
     *The data buffer can hold up to 62 disk blocks."
      IF (MODE.EQ.LONG) TYPE "(CR)
     *The data buffer can hold up to 200 disk blocks,(CR)
     *and are output in 40 block multiples."
C
C     The variable STBLK maintains the starting block position for each
C     write operation into the data buffer. The variable BLKLEFT maintains
C     the number of available blocks left in the data buffer that can
C     receive data.
C
      STBLK=0                          ;initialize data buffer
      IF (MODE.EQ.SHORT) BLKLEFT=62    ;write parameters
      IF (MODE.EQ.LONG) BLKLEFT=200

      CALL DFILW("DIGI.DT",IER)      ;erase the current data buffer to
      CALL CFILW("DIGI.DT",2,IER)    ;ready for new data
      DTBUF=EMPTY

 403  ACCEPT "(CR)
     *Enter the filename for reading:"
      READ(11,2) FILENAM(1)
 2    FORMAT (S13)

      CALL STAT(FILENAM,STATUS,IER)
      IF (IER.EQ.13) GO TO 420
      IF (IER.NE.1) TYPE "STAT error ",IER," with your file"
      FBLKS=STATUS(9)                          ;FBLKS is the number of full
      IF (STATUS(10).EQ.512) FBLKS=FBLKS+1     ;disk blocks in the user file
      IF (FBLKS.LT.1) GO TO 425

 405  WRITE(10,5) FILENAM(1),FBLKS,FBLKS
 5    FORMAT("(CR)
     *File ",S13," contains ",I3," disk blocks, numbered 1 - ",I3)

      ACCEPT "(CR)
     *Please specify the blocks to be read in,(CR)
     *first block: ",FIRBLK
      IF (FIRBLK.LT.1 .OR. FIRBLK.GT.FBLKS) GO TO 435
      ACCEPT "
     *last block: ",NUMBLK
      IF (NUMBLK.GT.FBLKS .OR. NUMBLK.LT.FIRBLK) GO TO 435
```

```
      BLKLEFT=BLKLEFT-((NUMBLK-FIRBLK)+1)
      IF (BLKLEFT.LT.0) GO TO 430
      FIRBLK=FIRBLK-1

      CALL OPEN(1,FILENAM,1,IER)
      IF (IER.NE.1) TYPE "OPEN error ",IER," with your file"
      CALL OPEN(2,"DIGI.DT",0,IER)
      IF (IER.NE.1) TYPE "OPEN error ",IER," with the data buffer"

      STBLK=STBLK+FIRBLK
415   READBLK=NUMBLK
      IF (READBLK.GT.62) READBLK=62
      NUMBLK=NUMBLK-READBLK

      CALL RDBLK(1,FIRBLK,WORK,READBLK,IER)
      IF (IER.NE.1) TYPE "RDBLK error ",IER," with your file"
      CALL WRBLK(2,STBLK,WORK,READBLK,IER)
      IF (IER.NE.1) TYPE "WRBLK error ",IER," with data buffer"
      FIRBLK=FIRBLK+READBLK
      STBLK=STBLK+READBLK
      IF (NUMBLK.NE.0) GO TO 415

      CALL RESET
      DTBUF=FULL    ;the data buffer contains data
      IF (BLKLEFT.EQ.0) GO TO 50    ;return to the main menu

416   WRITE(10,3) BLKLEFT
3     FORMAT("(CR)
     *The data buffer can hold ",I3," additional disk blocks.")
      ACCEPT "(CR)
     *Do you want to read from file into these blocks,(CR)
     *   1: yes(CR)
     *   2: no(CR)
     *selection:",MENU4
      IF (MENU4.EQ.1) GO TO 403
      IF (MENU4.EQ.2) GO TO 50
      WRITE(10,1)
      GO TO 416

420   TYPE "(CR)(7)(7)(7)
     *This file does not exist in the current directory."
      GO TO 450

425   TYPE "(CR)(7)(7)(7)
     *This file is less than one disk block."
      GO TO 450

430   BLKLEFT=BLKLEFT+((NUMBLK-FIRBLK)+1)
      WRITE(10,7) BLKLEFT
7     FORMAT("(CR)(7)(7)(7)
     *You specified too many blocks. The data buffer can(CR)
     *only hold ",I3," disk blocks. Please try again.")
      GO TO 405
```

```
435  TYPE "(CR)(7)(7)(7)
    *You cannot make that block(CR)
    *selection. Please try again."
     GO TO 405

450  ACCEPT "
    *Do you want to,(CR)
    *   1: try another file(CR)
    *   2: return to the main menu(CR)
    *selection:",MENU2

     IF (MENU2.EQ.1) GO TO 403
     IF (MENU2.EQ.2) GO TO 50
     WRITE (10,1)
     GO TO 450

C*************************************************************************

C
C     This section of code allows the user to write the data buffer
C     to a disk file at a specified beginning block number.
C
 700  IF (DTBUF.EQ.EMPTY) GO TO 251

      CALL STAT("DIGI.DT",STATUS,IER)
      DBLKS=STATUS(9)+1   ;DBLKS is the number of data buffer disk blocks

      WRITE(10,13) DBLKS,DBLKS
 13   FORMAT("(CR)
     *The data buffer contains ",I3," diskblks numbered 1 - ",I3,".")
      TYPE "
     *It can be written to specified blocks of an existing file,(CR)
     *or to a new file."

 705  ACCEPT "(CR)
     *Enter the filename for writing:"
      READ(11,11) FILENAM(1)
 11   FORMAT(S13)

      MENU3=0
      CALL STAT(FILENAM,STATUS,IER)
      IF (IER.EQ.13) GO TO 710   ;if the file does not exist, create it
      IF (IER.NE.1) TYPE "STAT error ",IER," with your file"

      FBLKS=STATUS(9)                      ;FBLKS is the number of full
      IF (STATUS(10).EQ.512) FBLKS=FBLKS+1 ;disk blocks in the user file

      GO TO 720

 710  CALL CFILW(FILENAM,2,IER)   ;create the user file as a random file
      IF (IER.NE.1) TYPE "CFILW error ",IER," with your file"

 715  CALL OPEN(1,"DIGI.DT",1,IER)
```

```
      IF (IER.NE.1) TYPE "OPEN error ",IER," with the data buffer"
      CALL OPEN(2,FILENAM,3,IER)
      IF (IER.NE.1) TYPE "OPEN error ",IER," with your file"

      STBLK=0
      FIRBLK=0
      WBLKS=DBLKS
      IF (MENU3.EQ.0 .OR. MENU3.EQ.1) GO TO 717

716   IF (FBLKS.NE.0) WRITE(10,15) FILENAM(1),FBLKS
15    FORMAT("(CR)
     *File ",S13," contains ",I3," disk blocks.")
      IF (FBLKS.EQ.0) WRITE(10,16) FILENAM(1)
16    FORMAT("(CR)
     *File ",S13," is empty, it contains zero disk blocks.")
      TYPE "
     *Disk blocks are numbered beginning with one."
      ACCEPT "(CR)
     *Please specify the starting block for the data buffer(CR)
     *to be written:",FIRBLK
      FIRBLK=FIRBLK-1
      IF (FIRBLK.GE.0 .AND. FIRBLK.LE.FBLKS) GO TO 717

      TYPE "(CR)(7)(7)(7)
     *You cannot make that block(CR)
     *selection. Please try again."
      GO TO 716

717   NUMBLK=WBLKS
      IF (NUMBLK.GT.62) NUMBLK=62
      WBLKS=WBLKS-NUMBLK
      CALL RDBLK(1,STBLK,WORK,NUMBLK,IER)
      IF (IER.NE.1) TYPE "RDBLK error ",IER," with the data buffer"
      CALL WRBLK(2,FIRBLK,WORK,NUMBLK,IER)
      IF (IER.NE.1) TYPE "WRBLK error ",IER," with your file"
      STBLK=STBLK+NUMBLK
      FIRBLK=FIRBLK+NUMBLK
      IF (WBLKS.NE.0) GO TO 717

      CALL RESET
      WRITE(10,12) FILENAM(1)
12    FORMAT("(CR)
     *The data buffer has been written to file ",S13)
      GO TO 50

720   WRITE(10,14) FBLKS
14    FORMAT("(CR)
     *This file already exists in the current directory.(CR)
     *It contains ",I3," disk blocks.")
      ACCEPT "(CR)
     *Do you want to,(CR)
     *   1: delete the current file(CR)
     *   2: overwrite specified blocks of the current file(CR)
     *   3: select a different file(CR)
```

```
*   4: return to the main menu(CR)
*selection:",MENU3
 CALL NEWSCR

 IF (MENU3.EQ.1) GO TO 730
 IF (MENU3.EQ.2) GO TO 715
 IF (MENU3.EQ.3) GO TO 705
 IF (MENU3.EQ.4) GO TO 50
 WRITE(10,1)
 GO TO 720

730  CALL DFILW(FILENAM,IER)
 IF (IER.NE.1) TYPE "DFILW error ",IER," with your file"
 GO TO 710


C*************************************************************************

C
C     This section of code copies the data buffer to the edit buffer.
C
600  IF (DTBUF.EQ.EMPTY) GO TO 251

 CALL STAT("DIGI.DT",STATUS,IER)
 BLKS=STATUS(9)+1

 CALL FOPEN(1,"DIGI.DT")
 CALL FOPEN(2,"DIGI.ED")

 FIRBLK=0
605  NUMBLK=BLKS
 IF (NUMBLK.GT.62) NUMBLK=62
 BLKS=BLKS-NUMBLK

 CALL RDBLK(1,FIRBLK,WORK,NUMBLK,IER)
 CALL WRBLK(2,FIRBLK,WORK,NUMBLK,IER)
 FIRBLK=FIRBLK+NUMBLK
 IF (BLKS.NE.0) GO TO 605

 CALL FCLOSE(1)
 CALL FCLOSE(2)

 EDBUF=FULL   ;the edit buffer contains data

 TYPE "(CR)
*The data buffer has been copied to the edit buffer."
 GO TO 50

C*************************************************************************

1000 CALL DFILW("DIGI.DT",IER)
 CALL DFILW("DIGI.ED",IER)
 CALL DFILW("DIGI.OU",IER)
 CALL EXIT
```

```
      END
C*************************************************************************
```

```
C*******************************************************************************

C     Title: Editor
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program handles the editing operations in the SPEECH package.
C        The SPEECH package is a six-program package that utilizes the
C        Eclipse A/D/A device to work with speech data files. This program
C        is not a stand-alone program. It's operation depends upon parameter
C        files created by other programs. To understand the operation of
C        this program, program SPEECH, which is the central program of the
C        package, should be consulted first.

C     Compile command:
C        FORTRAN EDITOR

C     Load command:
C        RLDR/P EDITOR NEWSCR @FLIB@

C     Comments:
C        The save file (.SV) of the following programs are required in
C        the user's directory to operate this package,

C             . SPEECH,EDITOR,SMALLIN,SMALLOUT,BIGIN,BIGOUT

C*******************************************************************************

      INTEGER STATUS(18),EBLKS,FIRST,LAST,INFO(2)
      INTEGER HBLKS,BBLKS(10),CLIPS(200),SAMPS(200,20),START,FIN
      INTEGER IMIN,IMAX,LEN,MORE,BFLAG
      INTEGER IBOT,ITOP,IOP,LIMIT,NUMBLK,FIRBLK
      INTEGER FILENAM(7),IDATA3(10240),FBLKS
      INTEGER STBLK,READBLK,MODE,SHORT,LONG,MENU1,MENU2

      REAL RMAXS(200),RMINS(200),RNUM,TOP,BOT,RMAX,RMIN,MAG(10)
      REAL INCR(20),TINCR,POINTS,CLIP

      DATA SHORT,LONG / 1,2 /

      IDATA1=64000K
      CALL NEWSCR
C
C     Retrieve variable MODE to determine which program to call to output
C     the edit buffer.
C
      CALL FOPEN(1,"DIGI.OU")
      CALL RDSEQ(1,MODE,2,IER)
      CALL FCLOSE(1)

C*******************************************************************************

  100 TYPE "(CR)
```

```
     *   --) the program is in edit mode (--"
      CALL STAT("DIGI.ED",STATUS,IER)   ;get the edit buffer's size
      EBLKS=STATUS(9)+1                 ;this is the number of blocks
                                        ;in the edit buffer
C
C     The variables NUMBLK and FIRBLK are used to represent the number of
C     blocks and the first block, respectively, that will be involved in
C     an edit buffer D/A operation.
C
      NUMBLK=EBLKS    ;set the D/A parameters to output
      FIRBLK=0        ;the entire edit buffer

      BFLAG=0    ;this flag is set to one when the histogram
                 ;parameters have been collected
C
C     This is the editing program's main menu.
C
 105  ACCEPT "(CR)
     *Please select which operation will be performed,(CR)
     *   1: D/A conversion of edit buffer(CR)
     *   2: voltage histogram(CR)
     *   3: block histogram(CR)
     *   4: write edit buffer to file(CR)
     *   5: return to main menu(CR)
     *selection:",MENU1
      CALL NEWSCR

      IF (MENU1.EQ.1) GO TO 500
      IF (MENU1.EQ.2 .OR. MENU1.EQ.3) GO TO 200
      IF (MENU1.EQ.4) GO TO 700
      IF (MENU1.EQ.5) GO TO 1000
      WRITE(10,1)
 1    FORMAT("(CR)(7)(CR)(7)(CR)(7)
     *Please select only from options given.")
      GO TO 105

C*****************************************************************************

C
C     This section of code collects the histogram parameters. It is executed
C     only once each time the editing program is called.
C
 200  IF (EBLKS.EQ.0) GO TO 250
      IF (BFLAG.EQ.1) GO TO 220

      TYPE "(CR)
     *   --) histogram parameters being collected (--"

      DO 203 I=1,EBLKS
      CLIPS(I)=0
      RMAXS(I)=-5.
      RMINS(I)=5.
      DO 203 J=1,20
      SAMPS(I,J)=0
```

```
203   CONTINUE

      NUMBLK=EBLKS
      J=1
      FIRBLK=0
      LIMIT=0
      CALL FOPEN(1,"DIGI.ED")

204   READBLK=NUMBLK
      IF (READBLK.GT.40) READBLK=40
      NUMBLK=NUMBLK-READBLK
      CALL RDBLK(1,FIRBLK,IDATA3,READBLK,IER)
      IF (IER.NE.1) TYPE "RDBLK error ",IER," with edit buffer"

      LIMIT=LIMIT+READBLK
      FIRBLK=FIRBLK+READBLK
      START=1
      FIN=256
205   DO 210 I=START,FIN
      RNUM=FLOAT(IDATA3(I))/32768.*5.
      IF (RNUM.EQ.5. .OR. RNUM.EQ.-5.) CLIPS(J)=CLIPS(J)+1
      IF (RNUM.GT.RMAXS(J)) RMAXS(J)=RNUM
      IF (RNUM.LT.RMINS(J)) RMINS(J)=RNUM
      TOP=5.0
      BOT=4.5
      DO 210 K=1,20
      IF (RNUM.LE.TOP .AND. RNUM.GE.BOT) SAMPS(J,K)=SAMPS(J,K)+1
      TOP=TOP-.5
      BOT=BOT-.5
210   CONTINUE
      J=J+1
      START=START+256
      FIN=FIN+256
      IF (J.LE.LIMIT) GO TO 205
      IF (NUMBLK.NE.0) GO TO 204
      BFLAG=1
      CALL FCLOSE(1)
      GO TO 220

C*************************************************************************

C
C     This message is sent to the screen if the user attempts a buffer
C     operation when the buffer is empty.
C
250   TYPE "(CR)(7)(7)(7)
     *The edit buffer is currently empty."
      GO TO 105

C*************************************************************************

C
C     This section of code requests from the user which blocks of the
C     edit buffer will be in the histogram.
```

```
C
 219  WRITE(10,1)
 220  WRITE(10,3) EBLKS,EBLKS
 3    FORMAT ("(CR)
     *The edit buffer contains ",I3," disk blocks, numbered 1 - ",I3)
      TYPE "
     *Please specify which blocks will be in the histogram"

      ACCEPT "(CR)
     *first block: ",FIRST
      IF (FIRST.LT.1) GO TO 219

      ACCEPT "
     *last block: ",LAST
      IF (LAST.GT.EBLKS .OR. FIRST.GT.LAST) GO TO 219

      HBLKS=(LAST-FIRST)+1   ;this is the number of blocks in the histogram

      NUMBLK=HBLKS      ;set the D/A parameters to only output
      FIRBLK=FIRST-1    ;the histogram blocks

      POINTS=HBLKS*256.    ;this is the number of samples in the
                           ;histogram blocks

      IF (MENU1.EQ.3) GO TO 350    ;to give the block histogram
                                   ;else give the voltage histogram

C*******************************************************************************

C
C     This section of code prepares and displays the voltage histogram. It
C     scans the parameter variables for the histogram blocks.
C
      TYPE "(CR)
     *    --) voltage histogram being prepared (--"

      DO 301 I=1,20
      INCR(I)=0.
 301  CONTINUE
      CLIP=0.
      RMAX=-5.
      RMIN=5.

      DO 302 I=FIRST,LAST
      CLIP=CLIP+CLIPS(I)
      IF (RMAXS(I).GT.RMAX) RMAX=RMAXS(I)
      IF (RMINS(I).LT.RMIN) RMIN=RMINS(I)
      DO 302 J=1,20
      INCR(J)=INCR(J)+SAMPS(I,J)
 302  CONTINUE
      IMAX=INT(RMAX/5.*32768.)
      IMIN=INT(RMIN/5.*32768.)

 325  TYPE "
```

```
     #Voltage Histogram"
      WRITE(10,4) FIRST,LAST,POINTS,CLIP
4     FORMAT ("
     #blocks:",I3,"-",I3,"    total samples: ",F6.0,"    total clips: ",F6.0)
      WRITE(10,5) RMAX,IMAX,RMIN,IMIN
5     FORMAT ("
     #max voltage: ",F7.4,"(",I6,")(CR)
     #min voltage: ",F7.4,"(",I6,")")
      TYPE "(CR)
     #     Voltage     Positive     Negative      Total(CR)
     #    Magnitude    Samples      Samples      Samples(CR)"

      TOP=5.0
      BOT=4.5
      J=20
      DO 303 I=1,10
      TINCR=INCR(J)+INCR(I)
      WRITE(10,6) TOP,BOT,INCR(I),INCR(J),TINCR
6     FORMAT ("
     #     ",F3.1,"-",F3.1,"     ",F6.0,"       ",F6.0,"       ",F6.0)
      TOP=TOP-.5
      BOT=BOT-.5
      J=J-1
303   CONTINUE
      GO TO 400

C*************************************************************************************

C
C     This section of code prepares and displays the block histogram. It
C     scans the parameter variables for the edit buffer data blocks to
C     be included in the histogram.
C
350   TYPE "(CR)
     #   --) block histogram being prepared (--"

      DO 351 I=1,10
      MAG(I)=-5.
      INCR(I)=0.
351   CONTINUE

      CLIP=0.
      RMAX=-.5
      RMIN=5.

      LEN=INT(HBLKS/10.)
      MORE=10-(HBLKS-(LEN*10))

      DO 352 I=1,10
      IF (LEN.NE.0) BBLKS(I)=LEN
      IF (LEN.EQ.0) BBLKS(I)=1
      IF (I.GT.MORE .AND. LEN.NE.0) BBLKS(I)=BBLKS(I)+1
      IF (LEN.EQ.0 .AND. I.GT.HBLKS) BBLKS(I)=0
352   CONTINUE
```

```
      J=1
      START=FIRST
354   FIN=START+BBLKS(J)-1
      DO 353 I=START,FIN
      CLIP=CLIP+CLIPS(I)
      INCR(J)=INCR(J)+CLIPS(I)
      IF (RMAXS(I).GT.RMAX) RMAX=RMAXS(I)
      IF (RMINS(I).LT.RMIN) RMIN=RMINS(I)
      IF (RMAXS(I).GE.MAG(J)) MAG(J)=RMAXS(I)
      IF (ABS(RMINS(I)).GE.MAG(J)) MAG(J)=ABS(RMINS(I))
353   CONTINUE
      J=J+1
      START=FIN+1
      IF (BBLKS(J).NE.0 .AND. J.LE.10) GO TO 354

      IMAX=INT(RMAX/5.*32768.)
      IMIN=INT(RMIN/5.*32768.)

365   TYPE "
     *Block Histogram"
      WRITE(10,4) FIRST,LAST,POINTS,CLIP
      WRITE(10,8) RMAX,IMAX,RMIN,IMIN
8     FORMAT("
     *max voltage: ",F7.4,"(",I6,")(CR)
     *min voltage: ",F7.4,"(",I6,")")
      TYPE "(CR)
     *    Block      Total      Max(CR)
     *    Number     Clips   Magnitude(CR)"

      I=0
      ITOP=FIRST
360   I=I+1
      IBOT=ITOP+BBLKS(I)-1
      IF (BBLKS(I).EQ.0) TYPE
      IF (BBLKS(I).EQ.1) WRITE(10,9) ITOP,INCR(I),MAG(I)
9     FORMAT("
     *     ",I3,"      ",F6.0,"      ",F7.4)
      IF (BBLKS(I).GT.1) WRITE(10,10) ITOP,IBOT,INCR(I),MAG(I)
10    FORMAT("
     *   ",I3,"-",I3,"    ",F6.0,"      ",F7.4)
      ITOP=IBOT+1
      IF (I.LT.10) GO TO 360
      GO TO 400

C*************************************************************************

C
C     This is the editing program's histogram menu.
C
400   ACCEPT "(CR)
     *Please select which operation will be performed,(CR)
     *   1: D/A conversion of histogram blocks(CR)
     *   2: delete histogram blocks from edit buffer(CR)
```

```
     *   3: return to the editing menu(CR)
     *selection:",MENU2
      CALL NEWSCR

      IF (MENU2.EQ.1) GO TO 500
      IF (MENU2.EQ.2) GO TO 600
      IF (MENU2.EQ.3) GO TO 650
      WRITE(10,1)
      GO TO 400

C*************************************************************************

 500  IF (EBLKS.EQ.0) GO TO 250

      TYPE "(CR)
     *   --> entering D/A mode <--"
C
C     Create a flag file which will indicate to the D/A program to output
C     the edit buffer instead of the data buffer. The parameters are
C     written to the flag file specifying the section of edit buffer to
C     output.
C
      INFO(1)=FIRBLK
      INFO(2)=NUMBLK
      CALL DFILW("DIGI.FG",IER)      ;delete possible flag file left from
                                     ;a previous abort
      CALL CFILW("DIGI.FG",2,IER)
      CALL FOPEN(1,"DIGI.FG")
      CALL WRSEQ(1,INFO,4,IER)
      CALL FCLOSE(1)

      IF (MODE.EQ.SHORT) CALL SWAP("SMALLOUT.SV",IER)
      IF (MODE.EQ.LONG) CALL SWAP("BIGOUT.SV",IER)
      IF (IER.NE.1) TYPE "SWAP error ",IER

      CALL DFILW("DIGI.FG",IER)
      IF (MENU1.EQ.1) GO TO 105    ;to the editing menu
      IF (MENU1.EQ.2) GO TO 325    ;to the volt hist menu
      IF (MENU1.EQ.3) GO TO 365    ;to the block hist menu

C*************************************************************************

C
C     This section of code deletes the histogram blocks by overwriting these
C     blocks in the edit buffer with the data immediately following the
C     histogram blocks. The histogram parameter arrays are similarily
C     updated.
C
 600  CALL FOPEN(1,"DIGI.ED")

      NUMBLK=EBLKS-LAST
      STBLK=LAST-1
 605  READBLK=NUMBLK
      IF (READBLK.GT.40) READBLK=40
```

```
      NUMBLK=NUMBLK-READBLK

606   CALL RDBLK(1,STBLK,IDATA3,READBLK,IER)
      IF (IER.NE.1) TYPE "RDBLK error ",IER," with edit buffer"
      CALL WRBLK(1,FIRBLK,IDATA3,READBLK,IER)
      IF (IER.NE.1) TYPE "WRBLK error ",IER," with edit buffer"
      STBLK=STBLK+READBLK
      FIRBLK=FIRBLK+READBLK
      IF (NUMBLK.NE.0) GO TO 605
      CALL FCLOSE(1)

      DO 615 I=FIRST,LAST
      J=I+HBLKS
      CLIPS(I)=CLIPS(J)
      RMAXS(I)=RMAXS(J)
      RMINS(I)=RMINS(J)
      DO 615 K=1,20
      SAMPS(I,K)=SAMPS(J,K)
615   CONTINUE

      EBLKS=EBLKS-HBLKS

      TYPE "(CR)
     *The edit buffer has been updated."

650   FIRBLK=0        ;set the D/A parameters to output
      NUMBLK=EBLKS    ;the entire edit buffer
      GO TO 105       ;return to main menu

C*********************************************************************************

700   IF (EBLKS.EQ.0) GO TO 250

      WRITE(10,13) EBLKS,EBLKS
13    FORMAT("(CR)
     *The edit buffer contains ",I3," diskblocks numbered 1 - ",I3,".")
      TYPE "
     *It can be written to specified blocks of an existing file,(CR)
     *or to a new file."

705   ACCEPT "(CR)
     *Enter the filename for writing:"
      READ(11,11) FILENAM(1)
11    FORMAT(S13)

      IOP=0
      CALL STAT(FILENAM,STATUS,IER)
      IF (IER.EQ.13) GO TO 710
      IF (IER.NE.1) TYPE "STAT error ",IER," with your file"
      FBLKS=STATUS(9)
      IF (STATUS(10).EQ.512) FBLKS=FBLKS+1
      GO TO 720

710   CALL CFILW(FILENAM,2,IER)
```

```
     IF (IER.NE.1) TYPE "CFILW error ",IER," with your file"

715  CALL OPEN(1,"DIGI.ED",1,IER)
     IF (IER.NE.1) TYPE "OPEN error ",IER," with the edit buffer"
     CALL OPEN(2,FILENAM,3,IER)
     IF (IER.NE.1) TYPE "OPEN error ",IER," with your file"

     STBLK=0
     FIRBLK=0
     BLKS=EBLKS
     IF (IOP.EQ.0 .OR. IOP.EQ.1) GO TO 717

716  IF (FBLKS.NE.0) WRITE(10,15) FILENAM(1),FBLKS
15   FORMAT("(CR)
    *File ",S13," contains ",I3," disk blocks.")
     IF (FBLKS.EQ.0) WRITE(10,16) FILENAM(1)
16   FORMAT("(CR)
    *File ",S13," is empty, it contains zero disk blocks.")
     TYPE "
    *Disk blocks are numbered beginning with one."
     ACCEPT "(CR)
    *Please specify the starting block for the data buffer(CR)
    *to be written:",FIRBLK
     FIRBLK=FIRBLK-1
     IF (FIRBLK.GE.0) GO TO 717
     GO TO 716

717  NUMBLK=BLKS
     IF (NUMBLK.GT.40) NUMBLK=40
     BLKS=BLKS-NUMBLK
     CALL RDBLK(1,STBLK,IDATA3,NUMBLK,IER)
     IF (IER.NE.1) TYPE "RDBLK error ",IER," with the data buffer"
     CALL WRBLK(2,FIRBLK,IDATA3,NUMBLK,IER)
     IF (IER.NE.1) TYPE "WRBLK error ",IER," with your file"
     STBLK=STBLK+NUMBLK
     FIRBLK=FIRBLK+NUMBLK
     IF (BLKS.NE.0) GO TO 717

     CALL RESET
     WRITE(10,12) FILENAM(1)
12   FORMAT("(CR)
    *The data buffer has been written to file ",S13)
     GO TO 105

720  WRITE(10,14) FBLKS
14   FORMAT("(CR)
    *This file already exists in the current directory.(CR)
    *It contains ",I3," disk blocks.")
     ACCEPT "(CR)
    *Do you want to,(CR)
    *    1: delete the current file(CR)
    *    2: overwrite specified blocks of the current file(CR)
    *    3: create a new file(CR)
    *    4: return to the editing menu(CR)
```

```
     *selection:",IOP
      CALL NEWSCR

      IF (IOP.EQ.1) GO TO 730
      IF (IOP.EQ.2) GO TO 715
      IF (IOP.EQ.3) GO TO 705
      IF (IOP.EQ.4) GO TO 105
      WRITE(10,1)
      GO TO 720

 730  CALL DFILW(FILENAM,IER)
      IF (IER.NE.1) TYPE "DFILW error ",IER," with your file"
      GO TO 710

C*******************************************************************************

 1000 CALL NEWSCR
      CALL EXIT
      END

C*******************************************************************************
```

```
C*****************************************************************************

C     Title: SmallIn
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program handles the short mode A/D conversion operations
C        in the SPEECH package. The SPEECH package is a six-program
C        package that utilizes the Eclipse A/D/A device to work with
C        speech data files. This program is not a stand-alone program.
C        It's operation depends upon parameter files created by other
C        programs. To understand the operation of this program, program
C        SPEECH, which is the central program of the package, should
C        be consulted first.

C     Compile command:
C        FORTRAN SMALLIN

C     Load command:
C        RLDR/P 2/K SMALLIN NEWSCR SAMCONFIG3 @SAMLIB@

C     Comments:
C        The save file (.SV) of the following programs are required in
C        the user's directory to operate this package,

C              SPEECH,EDITOR,SMALLIN,SMALLOUT,BIGIN,BIGOUT

C*****************************************************************************

      EXTERNAL IDS21   ;declare A/D device
      EXTERNAL IDS23   ;must also declare D/A device
      COMMON / IBUFF / IDATA3(15872)   ;setup the A/D conversion data buffer
      COMMON / IBUFO / IWASTE          ;must also set up a D/A buffer
      INTEGER IORBA(16)

      IDATA1=60000K   ;use channel one and external clock
      CALL NEWSCR   ;erase the screen

      TYPE "(CR)
     *   --)  the program is in A/D mode  (--"

      CALL DSTRT(IER)   ;initialize A/D/A device
      IF (IER.NE.1) CALL ERROR("DSTRT error")

      CALL OPEN(1,"DIGI.DT",3,IER)   ;ready program SPEECH's data
                                     ;buffer for writing
      IF (IER.NE.1) TYPE "OPEN error ",IER," with the data buffer"

  100 ACCEPT "(CR)
     *Press the carriage return to begin"
      ACCEPT

      CALL DOITW(IORBA,IDS21,8,IDATA1,15872,IDATA3,IER)
```

```
      IF (IER.NE.1) TYPE "DOITW error",IER

      TYPE "(7)(7)(7)(CR)
     *That's all folks!"

150   IOP=0
      ACCEPT "(CR)
     *press the carriage return to return to the main menu,(CR)
     *or press the space bar and carriage return to do a retake:"
      READ(11,2) IOP
2     FORMAT(S1)
      CALL NEWSCR

      IF (IOP.EQ.0) GO TO 200
      IF (IOP.EQ.8192) GO TO 100
      WRITE(10,1)
1     FORMAT("(CR)(7)(CR)(7)(CR)(7)
     *Please select only from options given.")
      GO TO 150

200   CALL WRBLK(1,0,IDATA3,62,IER)   ;write the conversion data to
                                      ;program SPEECH's data buffer
      IF (IER.NE.1) TYPE "WRBLK error ",IER,"with data buffer"
      CALL CLOSE(1,IER)
      IF (IER.NE.1) TYPE "CLOSE error",IER," with data buffer"

      TYPE "(CR)
     *   --) exiting A/D mode (--"
      CALL EXIT
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: SmallOut
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program handles the short mode D/A conversion operations
C        in the SPEECH package. The SPEECH package is a six-program
C        package that utilizes the Eclipse A/D/A device to work with
C        speech data files. This program is not a stand-alone program.
C        It's operation depends upon parameter files created by other
C        programs. To understand the operation of this program, program
C        SPEECH, which is the central program of the package, should
C        be consulted first.

C     Compile command:
C        FORTRAN SMALLOUT

C     Load command:
C        RLDR/P 2/K SMALLOUT NEWSCR SAMCONFIG4 @SAMLIB@

C     Comments:
C        The save file (.SV) of the following programs are required in
C        the user's directory to operate this package,

C             SPEECH,EDITOR,SMALLIN,SMALLOUT,BIGIN,BIGOUT

C*******************************************************************************

      EXTERNAL IDS21    ;must also declare A/D device
      EXTERNAL IDS23    ;declare D/A device
      COMMON / IBUFF / IWASTE          ;must also set up A/D buffer
      COMMON / IBUFD / IDATA3(15872)   ;set up the D/A conversion data buffer

      INTEGER IORBA(16),STATUS(18),NUMBLK,NUMB,FIRBLK,IEXT
      INTEGER PLACE,ZERO,INIT,READBLK,INFO(2)

      IDATA1=64000K   ;use channel one and external clock
      CALL NEWSCR   ;erase the screen

      TYPE "(CR)
     *   --) the program is in D/A mode (--"

      CALL DSTRT(IER)   ;initialize A/D/A device
      IF (IER.NE.1) CALL ERROR("DSTRT error")

      CALL OPEN(1,"DIGI.FG",1,IER)
      IF (IER.EQ.13) GO TO 100   ;if flag file does not exist, then
                                 ;output data buffer
      GO TO 200   ;else output the edit buffer

C*******************************************************************************
```

```
C
C     This section of code sets the output parameters to output the entire
C     data buffer.
C
 100  CALL CLOSE(1,IER)
      CALL STAT("DIGI.DT",STATUS,IER)
      NUMBLK=STATUS(9)+1
      FIRBLK=0
      CALL FOPEN(1,"DIGI.DT")   ;open the data buffer for reading
      GO TO 500

C*************************************************************************

C
C     This section of code retrieves the parameters from the flag file
C     that specifies the section of edit buffer to output.
C
 200  CALL RDSEQ(1,INFO,4,IER)
      CALL FCLOSE(1)

      FIRBLK=INFO(1)   ;the first block
      NUMBLK=INFO(2)   ;the number of data blocks
      CALL FOPEN(1,"DIGI.ED")   ;open the edit buffer for reading

C*************************************************************************

 500  CALL RDBLK(1,FIRBLK,IDATA3,NUMBLK,IER)
      IF (IER.NE.1) TYPE "RDBLK error ",IER," with output buffer"
      IDATA2=NUMBLK*256   ;the number of data samples to output

 505  ACCEPT "(CR)
     *Press the carriage return to begin"
      ACCEPT

 506  CALL DOITW(IORBA,IDS23,8,26624,IDATA2,IDATA3,IER)

      IF (IER.NE.1) TYPE "DOITW error ",IER

 510  IOP=0
      ACCEPT "(CR)
     *press carriage return to continue,(CR)
     *or press space bar and carriage return to repeat:"
      READ(11,2) IOP
 2    FORMAT(S1)
      CALL NEWSCR

      IF (IOP.EQ.0) GO TO 1000
      IF (IOP.EQ. 8192) GO TO 506
      WRITE(10,1)
 1    FORMAT("(CR)(7)(CR)(7)(CR)(7)
     *Please select options from the list only.")
      GO TO 510

 1000 TYPE "(CR)
```

```
*    --) exiting D/A mode (--"
     CALL EXIT
     END

C*************************************************************************************
```

```
C*************************************************************************

C     Title: BigIn
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program handles the long mode A/D conversion operations
C        in the SPEECH package. The SPEECH package is a six-program
C        package that utilizes the Eclipse A/D/A device to work with
C        speech data files. This program is not a stand-alone program.
C        It's operation depends upon parameter files created by other
C        programs. To understand the operation of this program, program
C        SPEECH, which is the central program of the package, should
C        be consulted first.

C     Compile command:
C        FORTRAN BIGIN

C     Load command:
C        RLDR/P 2/K 2000/N BIGIN NEWSCR SAMCONFIG3 @SAMLIB@

C     Comments:
C        The save file (.SV) of the following programs are required in
C        the user's directory to operate this package,

C              SPEECH,EDITOR,SMALLIN,SMALLOUT,BIGIN,BIGOUT

C*************************************************************************

      EXTERNAL IDS21   ;declare A/D device
      EXTERNAL IDS23   ;must also declare D/A device
      COMMON IWIND(10240)   ;extended memory window
      COMMON / IBUFF / IDATA3(10240)   ;set up the A/D conversion data buffer
      COMMON / IBUFO / IWASTE          ;must also set up a D/A buffer

      INTEGER IORBA(16),IFILE(7)

      CALL NEWSCR   ;erase the screen

      TYPE "(CR)
     #   --) the program is in A/D mode (--"
C
C     Set up extended memory to hold the results of the first four conversion
C     operation calls.
C
      CALL VMEM(IEXT,IER)
      IF (IER.NE.1) TYPE "VMEM error ",IER
      IF (IEXT.LT.40) CALL ERROR("insufficient extended memory")
      CALL MAPDF(40,IWIND,10,IER)   ;each conversion operation will
                                    ;collect 10KW of data
      IF (IER.NE.1) TYPE "MAPDF error ",IER

      CALL DSTRT(IER)   ;initialize A/D/A device
```

```
      IF (IER.NE.1) CALL ERROR("DSTRT error")
      CALL OPEN(1,"DIGI.DT",3,IER)    ;ready program SPEECH's data
                                      ;buffer for writing
      IF (IER.NE.1) TYPE "OPEN error ",IER," with data buffer"

  100 TYPE "(CR)
     #Press the carriage return to begin:"
      ACCEPT

      CALL DOITW(IORBA,IDS21,8,24576,10240,IDATA3,IER)
      CALL VSTASH(IDATA3,1,10240)

      CALL DOITW(IORBA,IDS21,8,24576,10240,IDATA3,IER)
      CALL VSTASH(IDATA3,10241,10240)

      CALL DOITW(IORBA,IDS21,8,24576,10240,IDATA3,IER)
      CALL VSTASH(IDATA3,20481,10240)

      CALL DOITW(IORBA,IDS21,8,24576,10240,IDATA3,IER)
      CALL VSTASH(IDATA3,30721,10240)

      CALL DOITW(IORBA,IDS21,8,24576,10240,IDATA3,IER)

      TYPE "(7)(7)(7)(CR)
     #That's all folks!"

  150 IOP=0
      ACCEPT "(CR)
     #Press carriage return to return to the main menu,(CR)
     #or press space bar and carriage return to do a retake:"
      READ(11,2) IOP
    2 FORMAT(S1)
      CALL NEWSCR
      IF (IOP.EQ.0) GO TO 200
      IF (IOP.EQ.8192) GO TO 100
      WRITE(10,1)
    1 FORMAT("(CR)(7)(CR)(7)(CR)(7)
     #Please select only from options given.")
      GO TO 150
C
C     Write the conversion data to program SPEECH's data buffer.
C
  200 CALL WRBLK(1,160,IDATA3,40,IER)
      IF (IER.NE.1) TYPE "WRBLK error ",IER," with the data buffer"

      CALL VFETCH(IDATA3,1,10240)
      CALL WRBLK(1,0,IDATA3,40,IER)

      CALL VFETCH(IDATA3,10241,10240)
      CALL WRBLK(1,40,IDATA3,40,IER)

      CALL VFETCH(IDATA3,20481,10240)
      CALL WRBLK(1,80,IDATA3,40,IER)
```

```
      CALL VFETCH(IDATA3,30721,10240)
      CALL WRBLK(1,120,IDATA3,40,IER)

      IF (IER.NE.1) TYPE "WRBLK error ",IER," with the data buffer"
      CALL CLOSE(1,IER)
      IF (IER.NE.1) TYPE "CLOSE error ",IER," with the data buffer"

      TYPE "(CR)
     *   --) exiting A/D mode (--"

      CALL EXIT
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: BigOut
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program handles the long mode D/A conversion operations
C        in the SPEECH package. The SPEECH package is a six-program
C        package that utilizes the Eclipse A/D/A device to work with
C        speech data files. This program is not a stand-alone program.
C        It's operation depends upon parameter files created by other
C        programs. To understand the operation of this program, program
C        SPEECH, which is the central program of the package, should
C        be consulted first.

C     Compile command:
C       FORTRAN BIGOUT

C     Load command:
C        RLDR/P 2/K 2000/N BIGOUT NEWSCR SAMCONFIG4 @SAMLIB@

C     Comments:
C        The save file (.SV) of the following programs are required in
C        the user's directory to operate this package,

C              SPEECH,EDITOR,SMALLIN,SMALLOUT,BIGIN,BIGOUT

C*******************************************************************************

      EXTERNAL IDS21    ;must also declare A/D device
      EXTERNAL IDS23    ;declare D/A device
      COMMON IWIND(10240)    ;extended memory window
      COMMON / IBUFF / IWASTE          ;must also set up A/D buffer
      COMMON / IBUFO / IDATA3(10240)   ;set up the D/A conversion data buffer

      INTEGER IORBA(16),STATUS(18),NUMBLK,LASBLK,FIRBLK,IEXT
      INTEGER PLACE,ZERO,INIT,READBLK,INFO(2),MANY,SET

      IDATA1=64000K    ;use channel one and external clock
      CALL NEWSCR    ;erase the screen

      TYPE "(CR)
     *    --) the program is in D/A mode (--"
C
C     Set up extended memory to hold conversion data to be output.
C
      CALL VMEM(IEXT,IER)
      IF (IER.NE.1) TYPE "VMEM error ",IER
      IF (IEXT.LT.40) CALL ERROR("insufficient extended memory")
      CALL MAPDF(40,IWIND,10,IER)    ;each conversion operation will
                                     ;output 10KW of data
      IF (IER.NE.1) TYPE "MAPDF error ",IER
```

```
      CALL DSTRT(IER)   ;initialize A/D/A device
      IF (IER.NE.1) CALL ERROR("DSTRT error")

      CALL OPEN(1,"DIGI.FG",1,IER)
      IF (IER.EQ.13) GO TO 100   ;if flag file does not exist, then
                                 ;output data buffer
      GO TO 200   ;else output the edit buffer

C*************************************************************************

C
C     This section of code sets the parameters to output the entire
C     data buffer.
C
 100  CALL CLOSE(1,IER)
      CALL STAT("DIGI.DT",STATUS,IER)
      NUMBLK=STATUS(9)+1
      FIRBLK=0
      CALL FOPEN(1,"DIGI.DT")   ;open the data buffer for reading
      TYPE "(CR)
     *   --) the program is in D/A mode (--"
      GO TO 500

C*************************************************************************

C
C     This section of code retrieves the parameters from the flag file
C     that specifies the section of edit buffer to output.
C
 200  CALL RDSEQ(1,INFO,4,IER)
      CALL FCLOSE(1)

      FIRBLK=INFO(1)   ;the first block
      NUMBLK=INFO(2)   ;the number of blocks
      CALL FOPEN(1,"DIGI.ED")

C*************************************************************************

C
C     This section of code places the data to be ouput in the first NUMBLK
C     blocks of the conversion data buffer. Any remaining conversion data
C     buffer blocks are zero filled. Each D/A operation will output 200
C     disk blocks of data in 40-disk block sections.
C
 500  CONTINUE

      PLACE=1
      ZERO=0
      INIT=FIRBLK
      LASBLK=NUMBLK
      SET=0

      IF (LASBLK.LE.40) GO TO 502
      LASBLK=40
```

```
      FIRBLK=FIRBLK+40
      NUMBLK=NUMBLK-40

  501 READBLK=NUMBLK
      IF (READBLK.GT.40) READBLK=40
      NUMBLK=NUMBLK-READBLK

      CALL RDBLK(1,FIRBLK,IDATA3,READBLK,IER)
      IF (IER.NE.1) TYPE "RDBLK error ",IER," with the output buffer"
      MANY=READBLK*256
      IF (MANY.EQ.10240) GO TO 520
      MANY=MANY+1
      DO 530 I=MANY,10240
      IDATA3(I)=ZERO
  530 CONTINUE
  520 CALL VSTASH(IDATA3,PLACE,MANY)
      PLACE=PLACE+MANY
      FIRBLK=FIRBLK+READBLK
      SET=SET+1
      IF (NUMBLK.NE.0) GO TO 501

  502 IF (SET.EQ.4) GO TO 503
      DO 540 I=1,10240
      IDATA3(I)=ZERO
  540 CONTINUE
      PLACE=(SET*10240)+1
      CALL VSTASH(IDATA3,PLACE,10240)
      SET=SET+1
      GO TO 502

  503 CALL RDBLK(1,INIT,IDATA3,LASBLK,IER)
      IF (IER.NE.1) TYPE "RDBLK error ",IER," with output buffer"
      PLACE=(LASBLK*256)+1

  504 IF (PLACE.GT.10240) GO TO 505
      IDATA3(PLACE)=ZERO
      PLACE=PLACE+1
      GO TO 504

C*************************************************************************

  505 ACCEPT "(CR)
     *Press the carriage return to begin"
      ACCEPT

  506 CALL DOITW(IORBA,IDS23,8,26624,10240,IDATA3,IER)

      CALL VFETCH(IDATA3,1,10240)
      CALL DOITW(IORBA,IDS23,8,26624,10240,IDATA3,IER)

      CALL VFETCH(IDATA3,10241,10240)
      CALL DOITW(IORBA,IDS23,8,26624,10240,IDATA3,IER)

      CALL VFETCH(IDATA3,20481,10240)
```

```
      CALL DOITW(IORBA,IDS23,8,26624,10240,IDATA3,IER)

      CALL VFETCH(IDATA3,30721,10240)
      CALL DOITW(IORBA,IDS23,8,26624,10240,IDATA3,IER)
      IF (IER.NE.1) TYPE "DOITW error ",IER

510   IOP=0
      ACCEPT "(CR)
     *press carriage return to continue,(CR)
     *or press space bar and carriage return to repeat:"
      READ(11,2) IOP
2     FORMAT(S1)
      CALL NEWSCR

      IF (IOP.EQ.0) GO TO 1000
      IF (IOP.EQ. 8192) GO TO 503
      WRITE(10,1)
1     FORMAT("(CR)(7)(CR)(7)(CR)(7)
     *Please select options from the list only.")
      GO TO 510

1000  TYPE "(CR)
     *    --) exiting D/A mode (--"
      CALL EXIT
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: Digitize
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This is the central program of a three-program package that
C        interactively allows the user to set different operating features
C        of the Eclipse A/D/A device.

C     Environment:
C        This is a Fortran V program that has been designed to run on a
C        mapped-RDOS Eclipse S/250 minicomputer equipped with a model 4331
C        single board converter.

C     Compile command:
C        FORTRAN DIGITIZE

C     Load command:
C        RLDR/P DIGITIZE @FLIB@

C     Comments:
C        Refer to line 5 of the program text for information regarding
C        the data buffers in this package.

C        The save file (.SV) of the following programs are required in
C        the user's directory to operate this package,

C                     DIGITIZE,INDIGI,OUTDIGI

C*******************************************************************************

      INTEGER OPTION

 5    TYPE "(CR)
     *The A/D and D/A data buffers are separate with each dimensioned(CR)
     *to their maximum spec size of 16KW. Due to the data buffers being(CR)
     *this large, this program swaps to program INDIGI.SV for A/D(CR)
     *conversions and swaps to OUTDIGI.SV for D/A operations.(CR)(CR)
     *To output data collected in A/D mode, the A/D data buffer must(CR)
     *be written to a disk file while in A/D mode and then read into(CR)
     *the D/A data buffer after switching to D/A mode."

 10   ACCEPT "(CR)
     *Please enter which operation will be performed,(CR)
     *   1: A/D conversions(CR)
     *   2: D/A conversions(CR)
     *   3: exit(CR)
     *selection:",OPTION
      IF (OPTION.GT.1 .OR. OPTION.LT.3) GO TO 20
      TYPE "(CR)(CR)(CR)
     *Please select options from the list only."
      GO TO 10
```

```
 20   IF (OPTION.EQ.1) CALL SWAP("INDIGI.SV",IER)
      IF (OPTION.EQ.2) CALL SWAP("OUTDIGI.SV",IER)
      IF (OPTION.EQ.3) GO TO 900
      IF (IER.NE.1) TYPE "(CR)
     *SWAP error ",IER
      GO TO 10

 900  CALL EXIT
      END

C*************************************************************************
```

```
C*******************************************************************************

C     Title: OutDigi
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program handles the D/A mode options in the DIGITIZE
C        package. The DIGITIZE package is a three-program package that
C        interactively allows the user to select different operating
C        features of the Eclipse A/D/A device. This program, however,
C        can be operated as a stand alone program.

C     Environment:
C        This is a Fortran V program that has been designed to run
C        on a mapped-RDOS Eclipse S/250 minicomputer equipped with a
C        model 4331 single board converter.

C     Compile command:
C        FORTRAN OUTDIGI

C     Load command:
C        RLDR/P 2/K OUTDIGI CLKSET CHNSET CNVSET SEEIT PAPER^
C         REDBUF SETUP HEADER WRTBUF WARNNG SAMCONFIG4 @SAMLIB@

C     Comments:
C        Refer to line 10 of the program text for the menu options of
C        this program.

C        The save file (.SV) of the following programs are required in
C        the user's directory to operate the DIGITIZE package,

C                        DIGITIZE,INDIGI,OUTDIGI

C*******************************************************************************

      EXTERNAL IDS21   ;A/D device required by SAM
      EXTERNAL IDS23   ;D/A device
      COMMON / IBUFF / IWAST   ;A/D data buffer required by SAM
      COMMON / IBUFO / IDATA3(16384)   ;D/A data buffer
      INTEGER FILENAM(7),IORBA(16),CLOCK,FIRST,MODE,DEVICE

      CALL DSTRT(IER)   ;always initialize device
      IF (IER.NE.1) CALL ERRUR("DSTRT error(CR)program aborted")

      DEVICE=23
      TYPE "(CR)
     #Program OUTDIGI.SV executing ---) the device is in D/A mode"

 10   ACCEPT "(CR)
     #Please select which operation will be performed,(CR)
     *   1: conversions(CR)
     *   2: data buffer display(CR)
     *   3: data buffer print(CR)
```

```
*    4: data buffer write to file(CR)
*    5: read from file to data buffer(CR)
*    6: data buffer demultiplexing(CR)
*    7: exit(CR)
*selection:",IOP

 IF (IOP.EQ.1) GO TO 20
 IF (IOP.EQ.2) GO TO 40
 IF (IOP.EQ.3) GO TO 40
 IF (IOP.EQ.4) GO TO 80
 IF (IOP.EQ.5) GO TO 50
 IF (IOP.EQ.6) GO TO 60
 IF (IOP.EQ.7) GO TO 90
 WRITE (10,1)
 GO TO 10
1   FORMAT ("(CR)(CR)(CR)
*Please make selections only from the given options")

C*************************************************************************

20   CALL CLKSET(DEVICE,CLOCK)   ;set the clock

     CALL CHNSET(DEVICE,FIRST,MODE)   ;set the channel

     IDATA1=((CLOCK.OR.FIRST).OR.MODE).OR.4000K

25   ACCEPT "(CR)
*Do you wish to set the conversion count (1),(CR)
*or perform an error check (2)?",IERR

     IF (IERR.EQ.1) GO TO 30
     IF (IERR.EQ.2) GO TO 35
     WRITE(10,1)
     GO TO 25

C*************************************************************************

30   CALL CNVSET(IDATA2)   ;set the conversion count

     CALL WARNNG(CLOCK)   ;give warning message for clock set
     ACCEPT

     CALL DOITW(IORBA, IDS23, 8, IDATA1, IDATA2, IDATA3, IER)

     TYPE "(CR)
*Conversion operation completed"
     IF (IER.NE.1) TYPE "DOIT error ",IER
     IF (IORBA(14).NE.40000K) TYPE "IORBA(14) return ",IORBA(14)
     IF (IER.EQ.1 .AND. IORBA(14).EQ.40000K) TYPE "No errors reported"
     GO TO 10

C*************************************************************************

40   CALL SETUP(IFOR,IOP,ISTART,ISTOP)   ;get the parameters specifying
```

```
                                        ;the section of data buffer to be
                                        ;worked with.
C
C     Display te user requested section of data buffer.
C
      IF (IOP.EQ.2) CALL SEEIT(IFOR,ISTART,ISTOP,IDATA3,16384)
C
C     Print the header and the user requested section of data buffer.
C
      IF (IOP.EQ.3) CALL HEADER(DEVICE,FIRST,MODE,IDATA2,IER,IORBA,CLOCK)
      IF (IOP.EQ.3) CALL PAPER(IFOR,ISTART,ISTOP,IDATA3,16384)
      GO TO 10

C*************************************************************************

 35   INCREM=1000
      IDATA2=INCREM
 37   CALL DOITW (IORBA, IDS23, 8, IDATA1, IDATA2, IDATA3, IER)
      IF (IER.NE.1) GO TO 38
      IDATA2=IDATA2+INCREM
      IF (IDATA2.GT.16384) GO TO 38
      GO TO 37

 38   IDATA2=IDATA2-INCREM
      CALL DOITW (IORBA, IDS23, 8, IDATA1, IDATA2, IDATA3, IER)
      INCREM=INT(INCREM/10.0)
      IF (INCREM.NE.0) GO TO 37
      TYPE "(CR)
     *DOIT error ",IER,"(CR)
     *on conversion count ",IDATA2
      TYPE "(7)(7)(7)"
      GO TO 10

C*************************************************************************

 50   CALL REDBUF(IDATA3,16384)    ;let the user read specified sections
                                   ;of a file into the data buffer.
      GO TO 10

C*************************************************************************

 60   TYPE "(CR)
     *The data buffer will be demultiplexed by retrieving(CR)
     *every Nth point from a specified starting point.(CR)
     *(CR)
     *There will be N-1 data buffer points skipped(CR)
     *between each two demultiplexed points.(CR)
     *(CR)
     *The first data buffer point is numbered one."

      ACCEPT "(CR)
     *Please specify,(CR)
     *   N:",INTH
      ACCEPT "
```

```
     *   starting point:",IONE

      IF (INTH.LT.2 .OR. IONE.GT.16384) GO TO 60

      ISTOP=16384/INTH
      INTH=INTH
      J=IONE
      DO 65 I=1,ISTOP
      IDATA3(I)=IDATA3(J)
      J=J+INTH
 65   CONTINUE
      DO 66 I=ISTOP+1,16384
      IDATA3(I)=0
 66   CONTINUE
      GO TO 10

C*************************************************************************

 80   CALL WRTBUF(IDATA3,16384)    ;let the user write specified sections
                                   ;of data buffer to file.
      GO TO 10

C*************************************************************************

 90   CALL EXIT
      END

C*************************************************************************
```

```
C*************************************************************************

C     Title: InDigi
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program handles the A/D mode options in the DIGITIZE
C        package. The DIGITIZE package is a three-program package that
C        interactively allows the user to select different operating
C        features of the Eclipse A/D/A device. This program, however,
C        can be operated as a stand alone program.

C     Environment:
C        This is a Fortran V program that has been desigined to run
C        on a mapped-RDOS Eclipse S/250 minicomputer equipped with a
C        model 4331 single board converter.

C     Compile command:
C        FORTRAN INDIGI

C     Load command:
C        RLDR/P 2/K INDIGI CLKSET CHNSET CNVSET SEEIT PAPER^
C         SETUP HEADER WRTBUF WARNNG SAMCONFIG3 @SAMLIB@

C     Comments:
C        Refer to line 10 of the program text for the menu options of
C        this program.

C        The save file (.SV) of the following programs are required in
C        the user's directory to operate the DIGITIZE package,

C                        DIGITIZE,INDIGI,OUTDIGI

C*************************************************************************

      EXTERNAL IDS21   ;A/D device
      EXTERNAL IDS23   ;D/A device required by SAM
      COMMON / IBUFF / IDATA3(16384) ;A/D data buffer
      COMMON / IBUFO / IWAST         ;D/A data buffer required by SAM

      INTEGER IORBA(16),DEVICE,CLOCK,FIRST,LAST
      DEVICE=21

      TYPE "(CR)
     *Program INDIGI.SV executing ---) the device is in A/D mode"

      CALL DSTRT(IER)   ;always initialize device
      IF (IER.NE.1) CALL ERROR("DSTRT error")

 10   ACCEPT "(CR)
     *Please select which operation will be performed,(CR)
     *   1: conversions(CR)
     *   2: data buffer display(CR)
```

```
      *   3: data buffer print(CR)
      *   4: data buffer write to file(CR)
      *   5: exit(CR)
      *selection:",IOP

       IF (IOP.EQ.1) GO TO 20
       IF (IOP.EQ.2) GO TO 50
       IF (IOP.EQ.3) GO TO 50
       IF (IOP.EQ.4) GO TO 60
       IF (IOP.EQ.5) GO TO 80
       WRITE (10,1)
       GO TO 10
 1     FORMAT ("(CR)(CR)(CR)
      *Please make selections only from the given options")

C*************************************************************************

 20    CALL CLKSET(DEVICE,CLOCK)    ;set the clock

       CALL CHNSET(DEVICE,FIRST,LAST)    ;set the channel

       IDATA1=(CLOCK.OR.FIRST).OR.LAST

       TYPE "(CR)
      *The device may give an error for conversion counts(CR)
      *above 16073.(CR)
      *(CR)
      *The error check option will return the maximum error(CR)
      *free conversion count for the set up given. The conversion(CR)
      *clock must allow for repeated conversion calls when using(CR)
      *this option."

 30    ACCEPT "(CR)
      *Do you wish to,(CR)
      *   1: set the conversion count(CR)
      *   2: perform an error check(CR)
      *selection:",IERR

       IF (IERR.EQ.1) GO TO 40
       IF (IERR.EQ.2) GO TO 35
       WRITE(10,1)
       GO TO 30

C*************************************************************************

 35    INCREM=1000
       IDATA2=INCREM
 37    CALL DOITW(IORBA,IDS21,8,IDATA1,IDATA2,IDATA3,IER)
       IF (IER.NE.1) GO TO 38
       IDATA2=IDATA2+INCREM
       IF (IDATA2.GT.16384) GO TO 38
       GO TO 37

 38    IDATA2=IDATA2-INCREM
```

```
      CALL DOITW(IORBA,IDS21,8,IDATA1,IDATA2,IDATA3,IER)
      INCREM=INT(INCREM/10.0)
      IF (INCREM.NE.0) GO TO 37
      TYPE "(CR)
     *DOIT error ",IER,"(CR)
     *on conversion count ",IDATA2
      TYPE "(7)(7)(7)"
      GO TO 10

C*************************************************************************

 40   CALL CNVSET(IDATA2)    ;set the conversion count

      CALL WARNNG(CLOCK)     ;give warning message for clock set
      ACCEPT

      CALL DOITW(IORBA,IDS21,8,IDATA1,IDATA2,IDATA3,IER)

      TYPE "(7)(7)(7)(CR)
     *Conversion operation completed"
      IF (IER.NE.1) TYPE "DOIT error ",IER
      IF (IORBA(14).NE.40000K) TYPE "IORBA(14) return ",IORBA(14)
      IF (IER.EQ.1 .AND. IORBA(14).EQ.40000K) TYPE "No errors reported"
      GO TO 10

C*************************************************************************

 50   CALL SETUP(IFOR,IOP,ISTART,ISTOP)    ;get the parameters specifying
                                           ;the section of data buffer to be
                                           ;worked with.
C
C     Display the user requested section of data buffer.
C
      IF (IOP.EQ.2) CALL SEEIT(IFOR,ISTART,ISTOP,IDATA3,16384)
C
C     Print the header and the user requested section of data buffer.
C
      IF (IOP.EQ.3) CALL HEADER(DEVICE,FIRST,LAST,IDATA2,IER,IORBA,CLOCK)
      IF (IOP.EQ.3) CALL PAPER(IFOR,ISTART,ISTOP,IDATA3,16384)
      GO TO 10

C*************************************************************************

 60   CALL WRTBUF(IDATA3,16384)    ;let the user write specified sections
                                   ;of data buffer to file.
      GO TO 10

C*************************************************************************

 80   CALL EXIT
      END

C*************************************************************************
```

```
C*************************************************************************

C     Title: Cnvrt
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program converts conversion data from the format used
C        by the Eclipse A/D/A device into real number data and vice-versa.

C     Compile Command:
C        FORTRAN CNVRT

C     Load Command:
C        RLDR/P 2000/N CNVRT COMLN SORT2 STATUS DELCHC FILCHC @FLIB@

C     Environment:
C        This is a Fortran V program that has been designed to run on
C        a mapped-RDOS Eclpise S/250 minicomputer.

C     Command line:
C        CNVRT (/R or /I) input/I [/D] output/O

C        where "input" and "output" are any legal RDOS filenames.

C        Either the R or I switch must be attached to CNVRT. The R
C        switch denotes conversion from device format to real and
C        the I switch denotes conversion from real to device format.

C        The input and output filenames can be typed in any order,
C        however, the I switch should always be attached to the
C        input file and the O switch should always be attached to the
C        output file.

C        The D switch can only be attached to the input file, and
C        deletes the input file after the output file has been created.

C     Comments:
C        This program is designed for use with conversion data that was
C        collected with or will be output by the device set at an operating
C        range of +/- 5v.

C        The output file will be created as a random file. If
C        it already exists, the original file will be deleted first.

C        The input file cannot be larger than 32768 disk blocks. There
C        is not an error check for this condition.

C        This program requires 8K of extended memory.

C*************************************************************************

      COMMON IWAST(1024)        ;Min size window required for
                                ;extended memory setup
```

Appendix D

Source Code for Signal Processing Software

```
      RDISC=RDISC+WBLK
      IF (RSWIT) ILN=ILN-(WBLK*128.)
      IF (ISWIT) ILN=ILN-(WBLK*256.)
      IF (AGAIN.EQ.1) GO TO 10    ;if true, have not finished all
                                  ;input data

 30   CALL CLOSE(2,IER)
      IF (IER.NE.1) TYPE "CLOSE error",IER,"with output file"
      CALL APPEND(2,FILEO,2,IER)
      IF (IER.NE.1) TYPE "APPEND error",IER,"with output file"
      BYTS=ILN*4
      IF (ISWIT) BYTS=BYTS/2
      IF (RSWIT) START=(WBLK*128)+1
      IF (ISWIT) START=(WBLK*256)+1
      IF (RSWIT) CALL WRSEQ(2,REALNUM(START),BYTS,IER)
      IF (ISWIT) CALL WRSEQ(2,INTNUM(START),BYTS,IER)
      IF (IER.NE.1) TYPE "WRSEQ error",IER,"with output file"
      CALL RESET

C*************************************************************************
C
C     Handle the D switch option.
C
      CALL DELCHC(FILEI,FI)
 80   CALL EXIT
      END

C*************************************************************************
```

```
      IF (IER.NE.1) TYPE "OPEN error",IER,"with output file"
C
C     Set up extended memory to hold input data. The 8KW of extended
C     memory can hold 8192 integer elements (to be converted to 8192 real)
C     or 4096 real elements (to be converted to 4096 integer).
C
      CALL VMEM(EXTM,IER)
      IF (IER.NE.1) TYPE "VMEM error",IER
      IF (EXTM.LT.8) CALL ERROR("not enough extended memory")
      IF (RSWIT) CALL MAPDF(8,IWAST,1,IER)  ;retrieve one-word elements
      IF (ISWIT) CALL MAPDF(8,IWAST,1,2,IER);retrieve two-word elements
      IF (IER.NE.1) TYPE "MAPDF error",IER

C*********************************************************************
C
C     Compute the number of data elements.
C
      ILN=(IBLKS*256.)+(LASTBYT/2.)
      IF (ISWIT) ILN=(IBLKS*128.)+(LASTBYT/4.)
      IBLKS=IBLKS+1    ;may try to read past EOF
      AGAIN=1
C
C     Work with 32-block sections of input data. This fills the 8KW
C     partition of extended memory.
C
 10   READBLK=IBLKS
      IF (IBLKS.GT.32) READBLK=32     ; 32 * 256 = 8192 = 8KW of storage
      CALL ERDB(1,IDISC,0,READBLK,CHEC,IER)
      IF (IER.EQ.9) GO TO 12   ;ignore EOF error
      IF (IER.NE.1) TYPE "ERDB error",IER

 12   INDEX=0

 20   INDEX=INDEX+1
      IF (RSWIT) CALL IVF(IHOLD,INDEX)
      IF (ISWIT) CALL VF(RHOLD,INDEX)
      IF (RSWIT) REALNUM(INDEX)=FLOAT(IHOLD)/32768.*TOPVOLT
      IF (ISWIT) INTNUM(INDEX)=INT(RHOLD/TOPVOLT*32768.)


      IF (RSWIT.AND.(INDEX.EQ.8192)) GO TO 25
      IF (ISWIT.AND.(INDEX.EQ.4096)) GO TO 25
      IF (FLOAT(INDEX).LT.ILN) GO TO 20
      AGAIN=0

 25   WBLK=INT(INDEX/256)*2
      IF (ISWIT) WBLK=INT(INDEX/256)
      IF (RSWIT) CALL WRBLK(2,RDISC,REALNUM,WBLK,CHEC,IER)
      IF (ISWIT) CALL WRBLK(2,RDISC,INTNUM,WBLK,CHEC,IER)
      IF (IER.NE.1) TYPE "WRBLK error",IER,"with output file"
      DBLKS=INT(INDEX/256)
      IF (ISWIT) DBLKS=DBLKS*2
      IBLKS=IBLKS-DBLKS
      IDISC=IDISC+DBLKS
```

```
      REAL REALNUM(8192),TOPVOLT,ILN,RHOLD

      INTEGER FILEI(7),FILEO(7),RET,FI(2),FO(2),MS(2)
      INTEGER LASTBYT,INDEX,CHEC,IHOLD,OBLKS,IBLKS
      INTEGER INTNUM(4096)
      INTEGER READBLK,EXTM,BYTS,IDISC,RDISC,AGAIN,WBLK,START

      LOGICAL ITEST,RSWIT,ISWIT

      DATA TOPVOLT,FI,FO,RDISC,IDISC / 5.,6*0 /

C*********************************************************************
C
C     Retrieve command line files and verify two exist.
C
      CALL COMLN(RET,HOLD,FILEI,FILEO,HOLD,MS,FI,FO,HOLD)
      IF (RET.EQ.2) GO TO 3
      CALL ERROR ("The command line must contain two files.")
C
C     Determine which type of file the output file will be. Verify
C     that only the I or R switch was attached to CNVRT.
C
 3    RSWIT=ITEST(MS(2),14)   ;if true, real output file
      ISWIT=ITEST(MS(1),7)    ;if true, integer output file
      IF (ISWIT.AND..NOT.RSWIT) GO TO 4
      IF (RSWIT.AND..NOT.ISWIT) GO TO 4
      CALL ERROR("must include /R or /I switch")
C
C     Sort the files and verify the I and O switches.
C
 4    CALL SORT2(9,15,FILEI,FILEO,FI,FO)
C
C     Verify that the filenames are not identical.
C
      CALL FILCHC(FILEI,FILEO)
C
C     Verify that the input file exists and retrieve it's size.
C
      CALL STATUS(FILEI,IBLKS,LASTBYT)
C
C     Prepare the input file for reading.
C
      CALL OPEN(1,FILEI,2,IER)
      IF (IER.NE.1) TYPE "OPEN error",IER,"with input file"
C
C     Prepare the output file for writing.
C
      CALL DFILW(FILEO,IER)
      IF (IER.EQ.13) GO TO 5   ;IER=13 implies the file does not exist
      IF (IER.NE.1) TYPE "DFILW error",IER,"with output file"
 5    CALL CFILW(FILEO,2,IER)
      IF (IER.NE.1) TYPE "CFILW error",IER,"with output file"
      CALL OPEN(2,FILEO,2,IER)
```

```
C*************************************************************************

C     Title: Conv
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program convolves an input file with an impulse response
C        file. The filtered output is written to a separate file.
C        All file data types are treated as real.

C     Compile command:
C        FORTRAN/T CONV

C     Load command:
C        RLDR/P 2000/N CONV COMLN SORT3 STATUS LENCHC RDBYTS FILCHC^
C         APS.LB @FLIB@

C     Environment:
C        This is a Fortran V program that has been designed to run on a
C        mapped-RDOS Eclipse S/250 minicomputer equipped with a model
C        130 array processor.

C     Command line:
C        CONV input/I [/D] output/O filter/F [/D]

C        where "input","output" and "filter" are any legal RDOS filenames.

C        The input, output and filter filenames can be typed in any order,
C        however, the I switch should always be attached to the input
C        file, the O switch should always be attached to the output file,
C        and the F switch should always be attached to the filter file.

C        The D switch can only be attached to the input and filter files,
C        and deletes these files after the output file has been created.

C     Comments:
C        The output file will be created as a random file. If it already
C        exists, the original file will be deleted.

C        The filter file can be up to 512 points long.

C        The input file cannot be over 32767 disk blocks long. There is not
C        an error check for this condition.

C        The program will abort if the output filename given is the
C        same as the input or filter filenames.

C*************************************************************************

      REAL WORK(2048),DATA(10240),ILN,OLN
      INTEGER RET,SP,FILEI(7),FI(2),FILEO(7),FO(2),FILER(7),FR(2)
      INTEGER RLN,TLN,TLOC,START,FIN,TOTBYTS
      INTEGER FILL,READBK,IDISCBK,ODISCBK,IBLKS,OBLKS,CHEC,INIT
```

```
      INTEGER STEST,ELN,NUMBYTS,MAXREAD,ZERO,RBLKS,RBYTS
      INTEGER CB1(0:CBMAX),CB2(0:CBMAX),CB3(0:CBMAX)

      INCLUDE "ARRAYP:F5APS.FR"   ;must include in any AP program
C
C     Initialize the AP.
C
      CALL APINIT(NIL,NIL,IER)
      IF (IER.NE.1) CALL ERROR("APINIT errorr")
C
C     Retrieve command line files and verify three.
C
      CALL COMLN(RET,SP,FILEI,FILEO,FILER,SP,FI,FO,FR)
      IF (RET.NE.3) CALL ERROR("command line must specify three files")
C
C     Sort the files and verify the I, O, and F switches.
C
      CALL SORT3(9,15,6,FILEI,FILEO,FILER,FI,FO,FR)
C
C     Verify that the filter file exists and is of proper length.
C
      CALL STATUS(FILER,RBLKS,RBYTS)
      CALL LENCHC(FILER,RBLKS,RBYTS,1,512)

      RLN=(RBLKS*128)+(RBYTS/4)   ;filter response length
      TOTBYTS=RLN*4   ;number of byts in filter
C
C     Get the filter response data and load it into to the top of
C     AP memory.
C
      CALL RDBYTS(FILER,TOTBYTS,WORK,512)
      CALL CBSET(CB1,CBL,RLN,CBAX,0,CBAAMM,WORK,IER)
      IF (IER.NE.1) TYPE" CBSET error ",IER," loading filter"
      CALL VLDR(CB1)
C
C     Verify that the input file exists.
C
      CALL STATUS(FILEI,IBLKS,IBYTS)
C
C     Verify that the output filename is not the same as the input or
C     filter filenames.
C
      CALL FILCHC(FILEO,FILER)
      CALL FILCHC(FILEO,FILEI)
C
C     Ready the output file.
C
      CALL DFILW(FILEO,IER)
      CALL CFILW(FILEO,2,IER)
      IF (IER.NE.1) TYPE "CFILW error",IER,"with output file"
      CALL OPEN(2,FILEO,2,IER)
      IF (IER.NE.1) TYPE "OPEN error",IER,"with output file"
C
C     Gather parameters.
```

```
C
      ILN=(IBLKS*128.)+(IBYTS/4.)   ;length of input data
      IBLKS=IBLKS+1    ;number of input data disk blocks,
                       ;last block may not be full
      OLN=FLOAT(RLN)+ILN-1.   ;length of output data
      TLOC=2*RLN       ;AP memory location where input data can be loaded
      TLN=2048-TLOC    ;length of convolution input section
      FILL=TLN-RLN     ;length of zero fill for the first set of input data
C
C     Ready the input file.
C
      CALL OPEN(1,FILEI,1,IER)
      IF (IER.NE.1) TYPE "OPEN error",IER,"with input file"
C
C     Set the CBSET arrays for the current filter size.
C
      CALL CBSET(CB1,CBL,TLN,CBAX,TLOC,CBAAMM,WORK,IER)
      CALL CBSET(CB2,CBL,TLN,CBAX,TLOC,CBAY,0,CBAZ,RLN,CBIMM,RLN,IER)
      CALL CBSET(CB3,CBL,TLN,CBAZ,RLN,CBAAMM,WORK,IER)
C
C     Set counters.
C
      IDISCBK=0
      ODISCBK=0
      STEST=0
      ZERO=FILL
      MAXREAD=64
 10   DO 20 I=1,ZERO
      DATA(I)=0.
 20   CONTINUE

      IF (STEST.EQ.0) GO TO 25   ;On subsequent reloading of the DATA
      DO 24 L=START,FIN          ;array, overlapping data and data to
      DATA(I)=DATA(L)            ;close to the bottom of the array to
      I=I+1                      ;allow convolving is reloaded at the top.
 24   CONTINUE
 25   CONTINUE
C
C     Load up the DATA array. The DATA array is used to hold both,
C     input and output data by performing an in-place convolution
C     as the VCONRZ routine does. Each output data save section is
C     written to the DATA array overwriting the first RLN-1 points
C     of its input data.
C
      READBK=IBLKS
      IF (IBLKS.GT.MAXREAD) READBK=MAXREAD

      CALL RDBLK (1,IDISCBK,DATA(I),READBK,IER)
      IF (IER.NE.1) TYPE "RDBLK error",IER,"with input file"
      ELN=I+((READBK-1)*128)+(IBYTS/4)
      IF (READBK.EQ.MAXREAD) ELN=I+(MAXREAD*128)

      DO 40 I=ELN,10240   ;zero fill the rest of the DATA array so
      DATA(I)=0.0         ;the convolution can overrun.
```

```
40    CONTINUE

      J=FILL-RLN    ;starting index value for input data to be convolved
      K=0

80    DO 60 I=1,TLN
      J=J+1
      WORK(I)=DATA(J)
60    CONTINUE

      CALL VLDR (CB1)

      CALL VCONRZ(CB2)

      CALL VSTR(CB3)

      INIT=RLN+1
      DO 70 I=INIT,TLN
      K=K+1
      DATA(K)=WORK(I)
70    CONTINUE

      STEST=ELN-(J-RLN)
      IF (FLOAT(K).GE.OLN) GO TO 74
      IF (FLOAT(K+FILL).GE.OLN) GO TO 71
      IF (STEST.LT.TLN) GO TO 75
71    J=J-RLN
      GO TO 80

74    K=INT(OLN)
      STEST=TLN+1

75    OBLKS=INT(K/128)
      START=J-RLN-(K-(OBLKS*128))+2
      FIN=ELN-1
      CALL WRBLK (2, ODISCBK, DATA, OBLKS, IER)
      IF (IER.NE.1) TYPE "WRBLK error",IER,"with output file"
      CALL CLOSE (2, IER)
      IF (IER.NE.1) TYPE "CLOSE error",IER,"with output file"
      CALL APPEND(2,FILEO,2,IER)
      IF (IER.NE.1) TYPE "APPEND error",IER,"with output file"
      ODISCBK=ODISCBK+OBLKS
      IDISCBK=IDISCBK+READBK
      IBLKS=IBLKS-READBK
      ZERO=FILL-RLN
      MAXREAD=48
      OLN=OLN-(OBLKS*128.)
      IF (STEST.LT.TLN) GO TO 10

      START=(OBLKS*128)+1
      NUMBYTS=OLN*4
      CALL WRSEQ (2, DATA(START), NUMBYTS, IER)
      IF (IER.NE.1) TYPE "WRSEQ error",IER,"with output file"
      CALL RESET
```

```
 1000 CALL EXIT
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: FFT
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program computes either the 1024-point or 2048-point DFT
C        of an input file of real elements. The input file is augmented
C        with zeros as necessary to make the input file the correct
C        size. The results is a file of complex elements.

C     Compile command:
C        FORTRAN/T FFT

C     Load command:
C        RLDR/P 2000/N FFT COMLN SORT2 STATUS LENCHC TOFILE DELCHC^
C         APS.LB @FLIB@

C     Environment:
C        This is a Fortran V program that has been designed to run on a
C        mapped-RDOS Eclipse S/250 minicomputer equipped with a model
C        130 array processor.

C     Command line:
C        FFT [/S] input/I [/D] output/O

C        where "input" and "output" are any legal RDOS filenames.

C        The input and output filenames can be typed in any order,
C        however, the I switch should always be attached to the input
C        file and the O switch should always be attached to the output
C        file.

C        The D switch can only be attached to the input file, and deletes
C        the input file after the output file has been created.

C        The S switch denotes that the 1024-point DFT of the input file
C        will be computed, otherwise the 2048-point DFT is computed.

C*******************************************************************************

      INTEGER RET,SP,FILEI(7),FILEO(7),FI(2),FO(2),XLN,XBLKS,MS(2)
      INTEGER COUNT,CB1(0:CBMAX),CB2(0:CBMAX),NUMBLK,LASTBYT
      LOGICAL ITEST,SWIT,NOSWIT
      REAL WORK(2048),PLAY(2048),FIRST,LAST,XFORM(4096)
C
C     Map all of AP memory.
C
      COMMON / APMEM / WORK

      INCLUDE "ARRAYP:F5APS.FR"   ;must include in any AP program
C
C     Initialize the AP and set up mapping window.
```

```
C
      CALL APINIT(NIL,WORK,4,IER)
      IF (IER.NE.1) CALL ERROR("APINIT error")
      CALL APMAP(WORK,0,4,IER)
      IF (IER.NE.1) CALL ERROR("APMAP error")
C
C     Retrieve command line files and verify only two.
C
      CALL COMLN(RET,SP,FILEI,FILEO,SP,MS,FI,FO,SP)
      IF (RET.NE.2) CALL ERROR("command line must specify two files")
C
C     Determine which length DFT will be computed.
C
      SWIT=ITEST(MS(2),13)    ;if true, the 1024 point
      NOSWIT=.NOT.SWIT        ;if  true, the 2048 point
C
C     Sort the files and verify the I and O switches.
C
      CALL SORT2(9,15,FILEI,FILEO,FI,FO)
C
C     Verify that the input file exists and retrieve its size.
C
      CALL STATUS(FILEI,NUMBLK,LASTBYT)
      XBLKS=NUMBLK+1                  ;number of input file disk blocks
      XLN=(NUMBLK*128)+(LASTBYT/4)    ;number of real input file elements
C
C     Verify that the input file is the proper length.
C
      IF (NOSWIT) CALL LENCHC(FILEI,NUMBLK,LASTBYT,16,2048)
      IF (SWIT) CALL LENCHC(FILEI,NUMBLK,LASTBYT,16,1024)
C
C     Get the input file data.
C
      CALL OPEN(1,FILEI,1,IER)
      IF (IER.NE.1) TYPE "OPEN error ",IER," with input file"
      CALL RDBLK(1,0,WORK,XBLKS,COUNT,IER)
      IF (IER.EQ.9 .AND. COUNT.EQ.NUMBLK) GO TO 50
      IF (IER.NE.1) TYPE "RDBLK error ",IER," with input file"
 50   CALL FCLOSE(1)
C
C     Augment the input file with zeros.
C
      DO 60 I=XLN+1,2048
      WORK(I)=0.
 60   CONTINUE
C
C     Take the DFT.
C
      IF (NOSWIT) CALL CBSET(CB1,CBL,1024,CBAXC,WORK,CBCW,CWDFT,IER)
      IF (SWIT) CALL CBSET(CB1,CBL,512,CBAXC,WORK,CBCW,CWDFT,IER)
      IF (IER.NE.1) TYPE "CBSET error ",IER," on transform"
      CALL VFFTC(CB1)
      CALL VBRC(CB1)
      CALL VFFTR(CB1)
```

```
C
C     Arrange the AP DFT results into proper format.
C
      XFORM(1)=WORK(1)        ;get the first element
      XFORM(2)=0.

      IF (NOSWIT) GO TO 59

      XFORM(1025)=WORK(2)     ;get the middle element of matrix operation
      XFORM(1026)=0.
      DO 63 I=3,1024          ;get the first half of the matrix
      XFORM(I)=WORK(I)        ;operation
 63   CONTINUE

      K=2049                  ;get the second half of the matrix operation
      DO 64 I=3,1024,2
      J=I+1
      K=K-2
      XFORM(K)=WORK(I)
      K=K+1
      XFORM(K)=-1.*WORK(J)
      K=K-1
 64   CONTINUE
      GO TO 70

 59   XFORM(2049)=WORK(2)
      XFORM(2050)=0.
      DO 61 I=3,2048
      XFORM(I)=WORK(I)
 61   CONTINUE

      K=4097
      DO 62 I=3,2048,2
      J=I+1
      K=K-2
      XFORM(K)=WORK(I)
      K=K+1
      XFORM(K)=WORK(J)
      K=K-1
 62   CONTINUE
C
C     Write results to file.
C
 70   IF (NOSWIT) CALL TOFILE(FILEO,XFORM,4096)
      IF (SWIT) CALL TOFILE(FILEO,XFORM,2048)
C
C     Handle the D switch option.
C
      CALL DELCHC(FILEI,FI)

 80   CALL EXIT
      END

C*************************************************************************
```

```
C*******************************************************************************

C     Title: IFFT
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program computes either the 1024-point or the 2048-point
C        inverse DFT of an input file. The input file must contain complex
C        elements in rectangular format, that is X + jY. The result is
C        a file of real elements.

C     Compile command:
C        FORTRAN/T IFFT

C     Load command:
C        RLDR/P 2000/N IFFT COMLN SORT2 STATUS LENCHC INFILE^
C         TOFILE DELCHC APS.LB @FLIB@

C     Environment:
C        This is a Fortran V program that has been designed to run on a
C        mapped-RDOS Eclipse S/250 minicomputer equipped with a model
C        130 array processor.

C     Command line:
C        IFFT [/S] input/I [/D] output/O

C        where "input" and "output" are any legal RDOS filenames.

C        The input and output filenames can be typed in any order,
C        however, the I switch should always be attached to the input
C        file and the O switch should always be attached to the output
C        file.

C        The D switch can only be attached to the input file, and deletes
C        the input file after the output file has been created.

C        The S switch denotes that the 1024-point inverse DFT of the
C        input file will be computed. If it is not present, the 2048-point
C        inverse DFT is computed.

C*******************************************************************************

      INTEGER RET,SP,FILEI(7),FILEO(7),FI(2),FO(2),MS(2),LENGTH
      INTEGER CB1(0:CBMAX),CB2(0:CBMAX)
      LOGICAL ITEST,SWIT,NOSWIT
      REAL WORK(2048),PI,TABLE(0:511),XFORM(2176)
C
C     Map all of array processor memory.
C
      COMMON / APMEM / WORK

      INCLUDE "ARRAYP:F5APS.FR"    ;must include in any AP program
C
```

```
C     Initialize the AP and set up mapping window.
C
      CALL APINIT(NIL,WORK,4,IER)
      IF (IER.NE.1) CALL ERROR("APINIT error")
      CALL APMAP(WORK,0,4,IER)
      IF (IER.NE.1) CALL ERROR("APMAP error")
C
C     Retrieve command line files and verify only two.
C
      CALL COMLN(RET,SP,FILEI,FILEO,SP,MS,FI,FO,SP)
      IF (RET.NE.2) CALL ERROR("command line must specify two files")
C
C     Determine which length inverse DFT will be computed.
C
      SWIT=ITEST(MS(2),13)   ;if true, the 1024-point
      NOSWIT=.NOT.SWIT         ;if true, the 2048-point
C
C     Sort the files and verify the I and O switches.
C
      CALL SORT2(9,15,FILEI,FILEO,FI,FO)
C
C     Verify the input file exists and retrieve its size.
C
      CALL STATUS(FILEI,IBLKS,IBYTS)
C
C     Verify that the input file is the proper length.
C
      IF (SWIT) CALL LENCHC(FILEI,IBLKS,IBYTS,2048,2048)
      IF (NOSWIT) CALL LENCHC(FILEI,IBLKS,IBYTS,4096,4096)
C
C     Get the input file data.
C
      IF (SWIT) CALL INFILE(FILEI,0,9,XFORM,1152)   ;must read 9 disk
                                                    ;blocks to get 1025 points
      IF (NOSWIT) CALL INFILE(FILEI,0,17,XFORM,2176);must read 17 disk
                                                    ;blocks to get 2049 points
C
C     Set up the cosine table that is required for inverse DFT
C     operations of equal to or more than 1024 points. This table
C     could be set up identically for other inverse DFT operations.
C
      PI=4.*ATAN(1.)
      DO 55 I=0,511
      TABLE(I)=COS((2.*PI*FLOAT(I))/2048.)
 55   CONTINUE
C
C     Arrange the input data into AP inverse DFT format and provide
C     scaling.
C
      IF (NOSWIT) GO TO 60

      WORK(1)=XFORM(1)/512.
      WORK(2)=XFORM(1025)/512.
      DO 59 I=3,1024
```

```
      WORK(I)=XFORM(I)/512.
 59   CONTINUE
      GO TO 70

 60   WORK(1)=XFORM(1)/1024.
      WORK(2)=XFORM(2049)/1024.
      DO 61 I=3,2048
      WORK(I)=XFORM(I)/1024.
 61   CONTINUE
C
C     Set the length for inverse DFT operation.
C
 70   IF (NOSWIT) LENGTH=1024
      IF (SWIT) LENGTH=512
C
C     Take the inverse DFT.
C
      CALL CBSET(CB1,CBL,LENGTH,CBAXC,WORK,CBCW,CWIFTR,CBAAMM,TABLE,
     *CBERMASK,APMALLER,IER)
      IF (IER.NE.1) TYPE "CBSET error ",IER," on transform"
      CALL VFFTR(CB1)
      CALL CBSET(CB2,CBL,LENGTH,CBAXC,WORK,CBCW,CWIFTC,CBERMASK,APMALLER,IER)
      IF (IER.NE.1) TYPE "CBSET error ",IER," on complex"
      CALL VFFTC(CB2)
      CALL VBRC(CB1)
C
C     Write the results to file.
C
      IF (SWIT) CALL TOFILE(FILEO,WORK,1024)
      IF (NOSWIT) CALL TOFILE(FILEO,WORK,2048)
C
C     Handle the D switch option.
C
      CALL DELCHC(FILEI,FI)

 80   CALL EXIT
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: Mag
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program takes an input file of either 1024 or 2048 complex
C        elements and computes the corresponding magnitude file. The
C        result is a file of real elements.

C     Compile command:
C        FORTRAN/T MAG

C     Load command:
C        RLDR/P 2000/N MAG COMLN SORT2 STATUS LENCHC TOFILE DELCHC^
C         INFILE APS.LB @FLIB@

C     Environment:
C        This is a Fortran V program that has been designed to run on a
C        mapped-RDOS Eclipse S/250 minicomputer equipped with a model
C        130 array processor.

C     Command line:
C        MAG [/S] input/I [/D] output/O

C        where "input" and "output" are any legal RDOS filenames.

C        The input and output filenames can be typed in any order,
C        however, the I switch should always be attached to the input
C        file and the O switch should always be attached to the output
C        file.

C        The D switch can only be attached to the input file, and deletes
C        the input file after the output file has been created.

C        The S switch denotes that the input file contains 1024 complex
C        elements, otherwise 2048 complex elements are assumed.

C*******************************************************************************

      INTEGER RET,SP,FILEI(7),FILEO(7),FI(2),FO(2),MS(2)
      INTEGER CBI(0:CBMAX),IBLKS,IBYTS
      REAL WORK(2048),ANSW(4096)
      LOGICAL ITEST,SWIT,NOSWIT
C
C     Map all of AP memory.
C
      COMMON / APMEM / WORK

      INCLUDE "ARRAYP:F5APS.FR"   ;must include in any AP program
C
C     Initialize the AP and set up mapping window.
C
```

```
      CALL APINIT(NIL,WORK,4,IER)
      IF (IER.NE.1) CALL ERROR("APINIT error")
      CALL APMAP(WORK,0,4,IER)
      IF (IER.NE.1) CALL ERROR("APMAP error")
C
C     Retrieve command line files and verify only two.
C
      CALL CUMLN(RET,SP,FILEI,FILEO,SP,MS,FI,FO,SP)
      IF (RET.NE.2) CALL ERROR("command line must specify two files")
C
C     Determine which length magnitude will be computed.
C
      SWIT=ITEST(MS(2),13)    ;if true, the 1024 element
      NOSWIT=.NOT.SWIT        ;if true, the 2048 element
C
C     Sort the files and verify the I and O switches.
C
      CALL SORT2(9,15,FILEI,FILEO,FI,FO)
C
C     Verify that the input file exists and is the proper length.
C
      CALL STATUS(FILEI,IBLKS,IBYTS)
      IF (NOSWIT) CALL LENCHC(FILEI,IBLKS,IBYTS,4096,4096)
      IF (SWIT) CALL LENCHC(FILEI,IBLKS,IBYTS,2048,2048)

      CALL CBSET(CB1,CBL,1024,CBAZC,WORK,CBAXC,WORK,IER)
      IF (IER.NE.1) TYPE "CBSET error ",IER," with square"

      IF (SWIT) GO TO 55

      K=0
      DO 52 I=0,16,16
      CALL INFILE(FILEI,I,16,WORK,2048)   ;Get a section of input data.
      CALL VSMA(CB1)                      ;Compute the square of the
                                          ;magnitude.
      DO 52 J=1,1024
      K=K+1
      ANSW(K)=SQRT(WORK(J))               ;Take data out of window.
 52   CONTINUE                            ;Get next section.
      GO TO 70

 55   CALL INFILE(FILEI,0,16,WORK,2048)
      CALL VSMA(CB1)
C
C     Write results to file.
C
 70   IF (NOSWIT) CALL TOFILE(FILEO,ANSW,2048)
      IF (SWIT) CALL TOFILE(FILEO,WORK,1024)
C
C     Handle the D switch option.
C
      CALL DELCHC(FILEI,FI)

 80   CALL EXIT
```

```
      END
C*************************************************************************
```

```
C*******************************************************************************

C     Title: Mult
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program multiplies the individual elements of two 1024-point
C        or two 2048-point element files together to form a third file.
C        The two files must have matching data types (either real or
C        complex), which will be the data type of the third file.

C     Compile command:
C        FORTRAN/T MULT

C     Load command:
C        RLDR/P MULT COMLN SORT3 STATUS LENCHC INFILE TOFILE^
C         DELCHC APS.LB @FLIB@

C     Environment:
C        This is a Fortran V program that has been designed to run on a
C        mapped-RDOS Eclipse S/250 minicomputer equipped with a model
C        130 array processor.

C     Command line:
C        MULT [[/C or /R]/S] input/I [/D] output/O filter/F [/D]

C        where "input","output" and "filter" are any legal RDOS filenames.

C        The /C or /R switch must be included and signifies either complex
C        or real data files, respectively.

C        The S switch denotes that the input file contains 1024 elements.
C        If it is not present, it is assumed that there are 2048 elements
C        in the input file.

C        The input, output and filter filenames can be typed in any order,
C        however, the I switch should always be attached to the input
C        file, the O switch should always be attached to the output file,
C        and the F switch should always be attached to the filter file.

C        The D switch can only be attached to the input and filter
C        files, and deletes these files after the output file has
C        been created.

C*******************************************************************************

      INTEGER RET,SP,FILEI(7),FILEO(7),FILER(7),FI(2),FO(2),FR(2),MS(2)
      INTEGER CB1(0:CBMAX),IBLKS,IBYTS,RBLKS,RBYTS
      INTEGER STOP,SIZE
      REAL WORK(1024),PLAY(1024),ANSW(4096)
      LOGICAL ITEST,SWIT,NOSWIT
C
C     Map all of AP memory.
```

```
C
      COMMON / APMEM / WORK,PLAY

      INCLUDE "ARRAYP:F5APS.FR"   ;must include in any AP program
C
C     Initialize the AP and set up mapping window.
C
      CALL APINIT(NIL,WORK,4,IER)
      IF (IER.NE.1) CALL ERROR("APINIT error")
      CALL APMAP(WORK,0,4,IER)
      IF (IER.NE.1) CALL ERROR("APMAP error")
C
C     Retrieve command line files and verify three.
C
      CALL COMLN(RET,SP,FILEI,FILEO,FILER,MS,FI,FO,FR)
      IF (RET.NE.3) CALL ERROR("command line must specify three files")
C
C     Determine the element length of the input file.
C
      SWIT=ITEST(MS(2),13)   ;if true, 1024 elements
      NOSWIT=.NOT.SWIT       ;if true, 2048 elements
      IF (SWIT) SIZE=2048
      IF (NOSWIT) SIZE=4096
C
C     Sort the files and verify the I, O, and F switches.
C
      CALL SORT3(9,15,18,FILEI,FILEO,FILER,FI,FO,FR)
C
C     Verify that the input and filter files exist and retrieve their
C     size.
C
      CALL STATUS(FILEI,IBLKS,IBYTS)
      CALL STATUS(FILER,RBLKS,RBYTS)
C
C      Determine the type of data file elements.
C
      IF (ITEST(MS(1),11)) GO TO 50    ;complex data
      IF (ITEST(MS(1),3)) GO TO 60     ;real data
      CALL ERROR("program name must have either /C or /R attached")

C*************************************************************************

C
C     This section of code performs a complex multiplication.
C
 50   CALL CBSET(CB1,CBL,512,CBAZC,WORK,CBAXC,WORK,CBAYC,PLAY,IER)
      IF (IER.NE.1) TYPE "CBSET error ",IER," with complex"
C
C     Verify the input and filter file lengths for complex data.
C
      CALL LENCHC(FILEI,IBLKS,IBYTS,SIZE,SIZE)
      CALL LENCHC(FILER,RBLKS,RBYTS,SIZE,SIZE)

      IF (SWIT) STOP=8
```

```
      IF (NOSWIT) STOP=24

      K=0
      DO 52 I=0,STOP,8
      CALL INFILE(FILEI,I,8,WORK,1024)   ;Get input
      CALL INFILE(FILER,I,8,PLAY,1024)   ;and filter data
      CALL VMCA(CB1)                     ;and perform a complex multiplication.
      DO 52 J=1,1024
      K=K+1
      ANSW(K)=WORK(J)
 52   CONTINUE
C
C     Write results to file.
C
      CALL TOFILE(FILEO,ANSW,SIZE)
      GO TO 80

C*********************************************************************************

C
C     This section of code performs a real multiplication.
C
 60   CALL CBSET(CB1,CBL,1024,CBAZC,WORK,CBAXC,WORK,CBAYC,PLAY,IER)
      IF (IER.NE.1) TYPE "CBSET error ",IER," with real"
C
C     Verify the input and filter file lengths for real data.
C
      CALL LENCHC(FILEI,IBLKS,IBYTS,SIZE,SIZE)
      CALL LENCHC(FILER,RBLKS,RBYTS,SIZE,SIZE)

      IF (SWIT) STOP=0
      IF (NOSWIT) STOP=8

      K=0
      DO 62 I=0,STOP,8
      CALL INFILE(FILEI,I,8,WORK,1024)   ;Get input
      CALL INFILE(FILER,I,8,PLAY,1024)   ;and filter data
      CALL VMRA(CB1)                     ;and perform a real multiplication.
      DO 62 J=1,1024
      K=K+1
      ANSW(K)=WORK(J)
 62   CONTINUE
C
C     Write results to file.
C
      CALL TOFILE(FILEO,ANSW,SIZE)
      GO TO 80

C*********************************************************************************

C
C     Handle the D switch option.
C
 80   CALL DELCHC(FILEI,FI)
```

```
      CALL DELCHC(FILER,FR)

      CALL EXIT
      END

C*************************************************************************
```

Appendix E

User's Manual
and
Source Code
for
Filter Design Software

User's Manual for Interactive Filter Design with Program LPFIR

This user's manual explains how to adjust the filter design parameters in program LPFIR to obtain the filter that most closely approaches the user's specifications. It also explains how to set up a macro file to allow program LPFIR to be used with other programs to design filters in an interactive environment. This macro should be executed on a Tetronix graphics terminal interfaced with the Eclipse computer. The user should also verify that the array processor has been initialized.

Macro File Setup

The save file (.SV) of the following programs are required in the user's directory to execute the macro file,

LPFIR,FFT,MAG,FILTPLOT

The macro file can be built using the SPEED editor. The macro filename chosen by the user should be appended with the .MC extension when entering the editor. The following command would be used to enter the editor and build a macro file named FILTER.

SPEED FILTER.MC

Once in the editor, the user should insert the following character string with the I command.

```
LPFIR PFILE/P FFILE/F;FFT/S FFILE/I/D CFILE/O;
MAG/S CFILE/I/D LOGMAG/O;FILTPLOT/L LOGMAG
```

The user should refer to the source code heading of

each program for a description of the switch definitions and data files. After the above macro file has been executed, the user's directory will contain the following files,

PFILE	a parameter file describing the filter
LOGMAG	a file containing the magnitude of the 1024-point DFT of the filter impulse response

Macro Execution

The macro file can be executed by typing the macro filename with or without the .MC extension. Program LPFIR was not designed for use on a Tetronix graphics terminal. This section of the macro file can be executed on the Tetronix terminal, however, the screen must be manually erased when necessary by the user depressing the PAGE key. The user can also allow this section of the macro to be executed on the non-graphics terminal and then switch to the Tetronix terminal prior to execution of the plotting section of the macro.

Parameter Adjustment

The user must begin the parameter adjustment sequence with a filter design that does not generate a program error. To obtain an initial design, specify the desired filter with a set of fairly relaxed parameters. The following guide may be helpful.

1. small filter length, 20-50 points
2. large band lengths, .05-.10
3. large transition regions, .05-.10
4. low weight factors, 5-20

The grid density is a factor affecting the resolution of the filter, much like the filter length. It should be chosen to be 16 for the most resolution.

The deviation numbers that are displayed, while the filter is being designed, are an indication of how close the filter is approaching the design parameters. If the magnitude of the numbers remains less than 1, the design will generally be reasonable.

A design example will be given to clarify the design sequence. The example will design a notch filter to remove a tone located at .1 on the frequency scale. The initial design for such a filter is shown in Fig 1.

The filter length, with respect to being an odd or even number, appears to affect the program's ability to design a filter. The user can determine which type of filter length is best for the given design, by holding all other parameters constant and changing only the filter length. The result of doing this for the design example is shown in Fig 2. Since the odd filter length yielded the better design, all subsequent filter lengths will be odd.

Filters of larger lengths can have sharper transition regions and narrower bands. Therefore, the next steps involve increasing the filter's length until a filter with an accept-

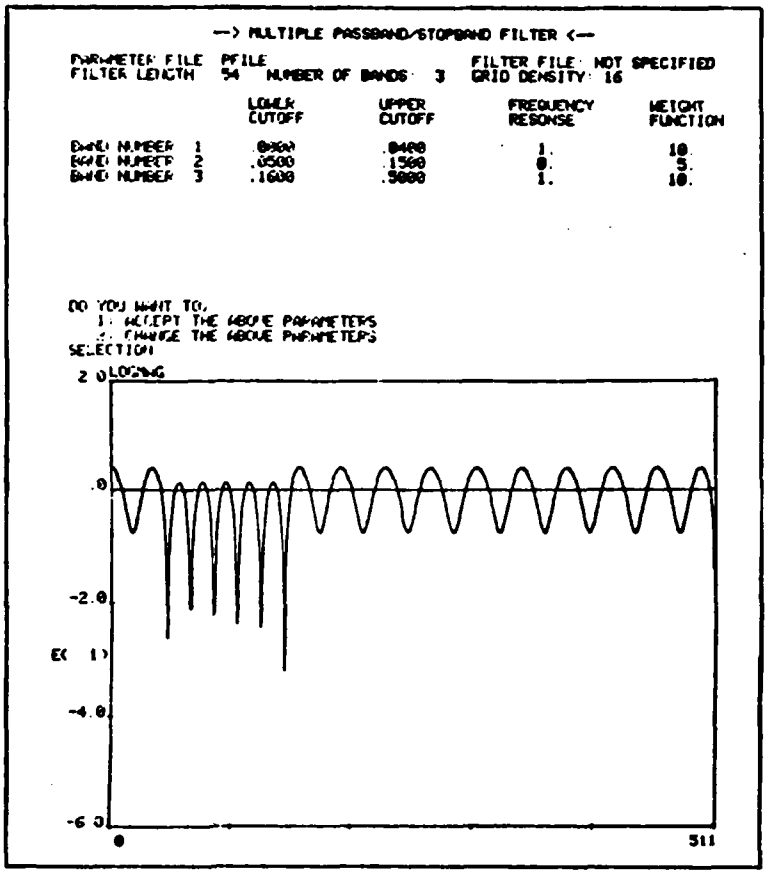

—> MULTIPLE PASSBAND/STOPBAND FILTER <—
PARAMETER FILE PFILE
FILTER LENGTH 54 NUMBER OF BANDS 3
FILTER FILE: NOT SPECIFIED
GRID DENSITY: 16
LOWER CUTOFF
UPPER CUTOFF
FREQUENCY RESONSE
WEIGHT FUNCTION
BAND NUMBER 1 .0000 .0400 1. 10.
BAND NUMBER 2 .0500 .1500 0. 5.
BAND NUMBER 3 .1600 .5000 1. 10.
DO YOU WANT TO,
1: ACCEPT THE ABOVE PARAMETERS
2: CHANGE THE ABOVE PARAMETERS
SELECTION:
LOG MAG
2.0
.0
-2.0
E(1)
-4.0
-6.0
0
511

Fig 1

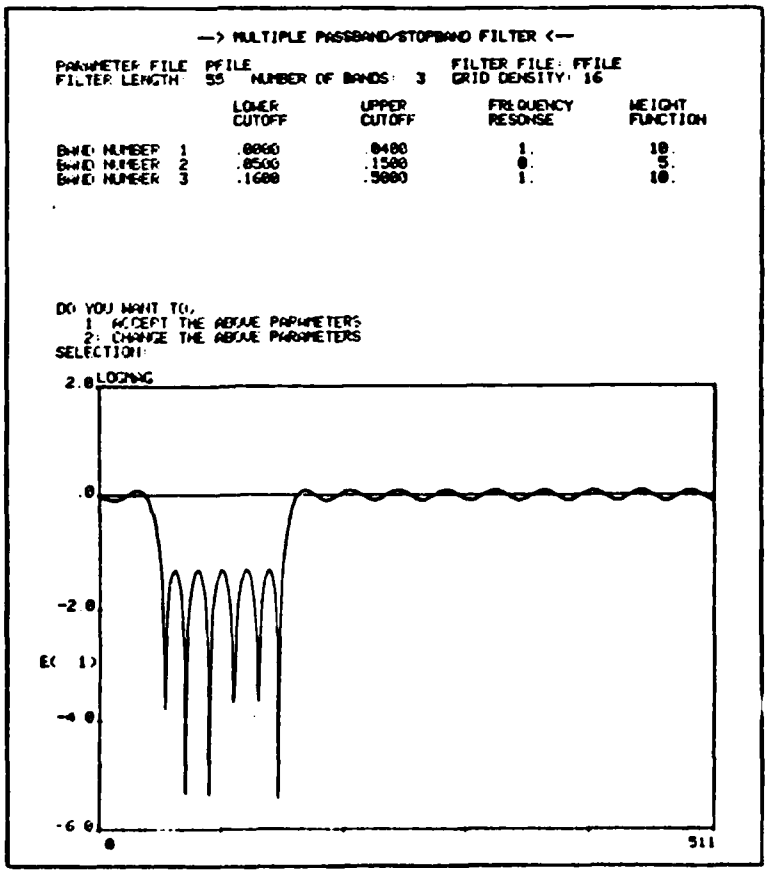

—> MULTIPLE PASSBAND/STOPBAND FILTER <—
PARAMETER FILE PFILE
FILTER LENGTH 55 NUMBER OF BANDS 3
FILTER FILE: PFILE
GRID DENSITY: 16
LOWER CUTOFF
UPPER CUTOFF
FREQUENCY RESONSE
WEIGHT FUNCTION
BAND NUMBER 1 .0000 .0400 1. 10.
BAND NUMBER 2 .0500 .1500 0. 5.
BAND NUMBER 3 .1600 .5000 1. 10.
DO YOU WANT TO,
1: ACCEPT THE ABOVE PARAMETERS
2: CHANGE THE ABOVE PARAMETERS
SELECTION:
LOG MAG
2.0
.0
-2.0
E(1)
-4.0
-6.0
0
511

Fig 2

able transition and bandwidth are obtained. For the design example, the filter length was increased to 165 points with the result shown in Fig 3. Since the ripple appears uneven, this filter was on the verge of not converging. The ripple should be improved before other parameters are varied. The filter length was reduced to 95 points with the results shown in Fig 4.

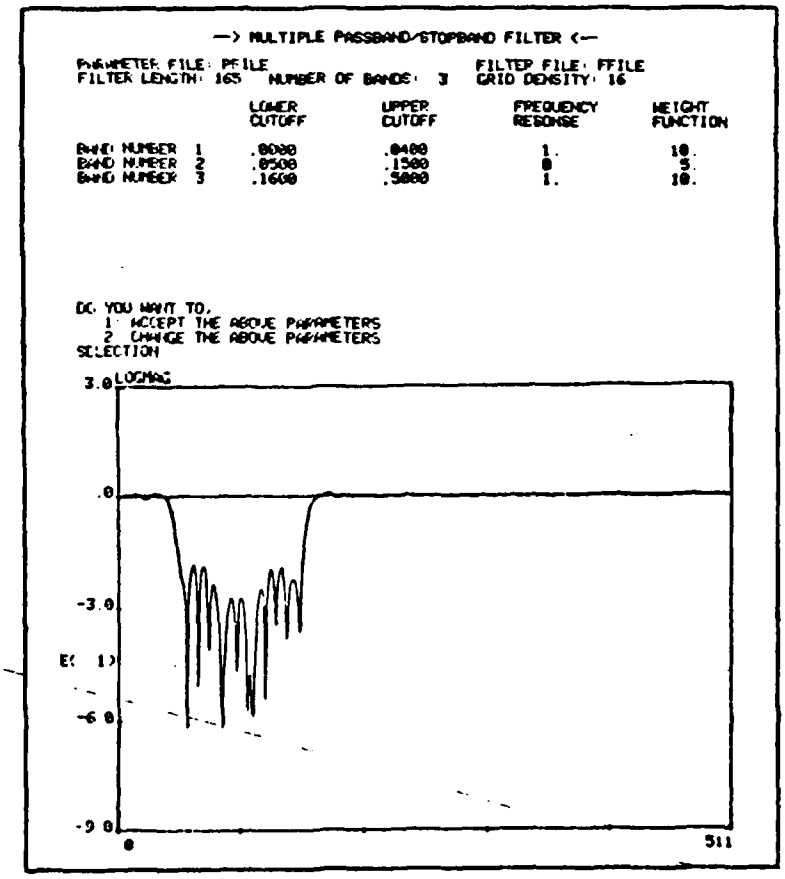


Fig 3

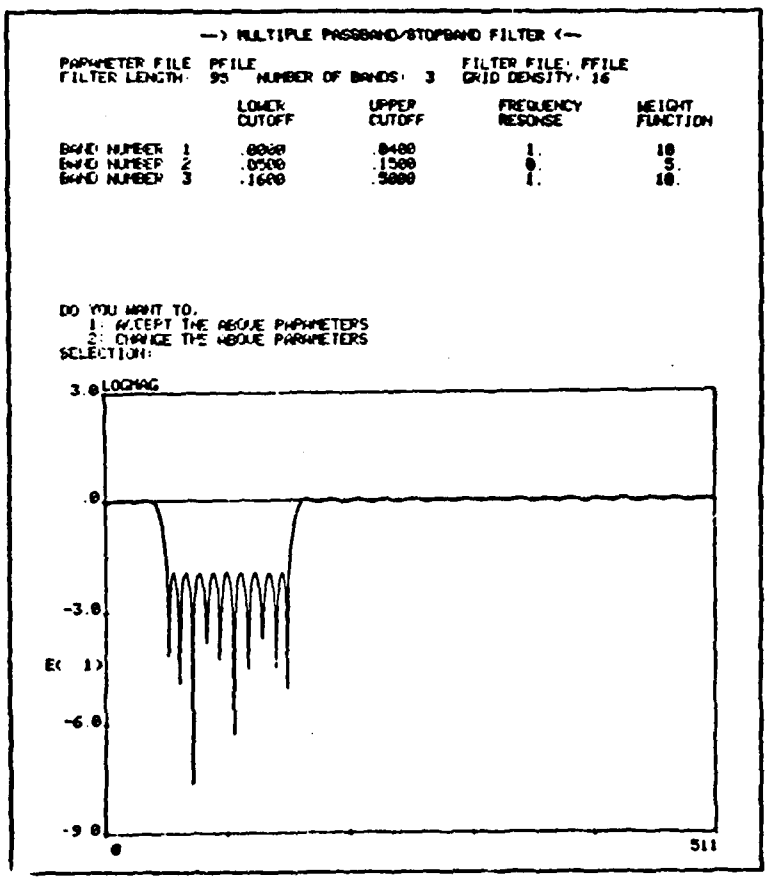

—) MULTIPLE PASSBAND/STOPBAND FILTER (—
PARAMETER FILE: PFILE
FILTER LENGTH: 95 NUMBER OF BANDS: 3
FILTER FILE: FFILE
GRID DENSITY: 16
LOWER CUTOFF
UPPER CUTOFF
FREQUENCY RESPONSE
WEIGHT FUNCTION
BAND NUMBER 1 .0000 .0400 1. 10
BAND NUMBER 2 .0500 .1500 0. 5.
BAND NUMBER 3 .1600 .5000 1. 10.
DO YOU WANT TO:
1: ACCEPT THE ABOVE PARAMETERS
2: CHANGE THE ABOVE PARAMETERS
SELECTION:
3.0
LOGMAG
.0
-3.0
E(1)
-6.0
-9.0
0
511

Fig 4

The number of ripples in the bandstop band indicates that the transition and bandstop width can be reduced. The final result of several attempts, that varied only the cutoff frequencies, is shown in Fig 5.

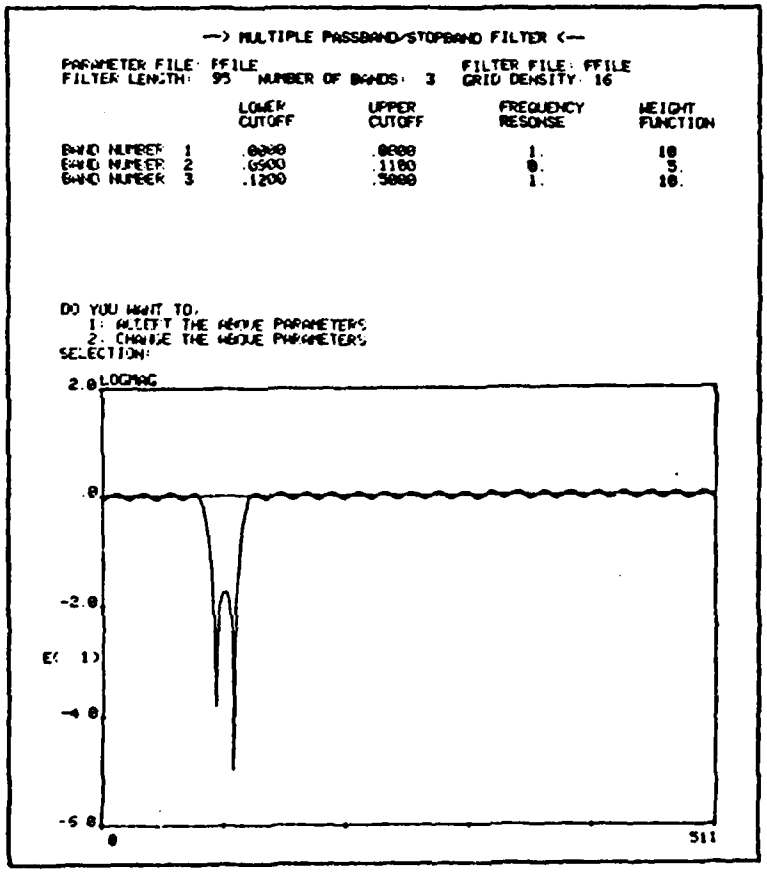


Fig 5

The stopband contains only one ripple. This indicates that the width of the stopband and transition regions cannot be decreased significantly without increasing the filter length. An attempt to do this without increasing the filter length caused the ripple to increase as shown in Fig 6. The result of increasing the filter length only gave much better results as shown in Fig 7.

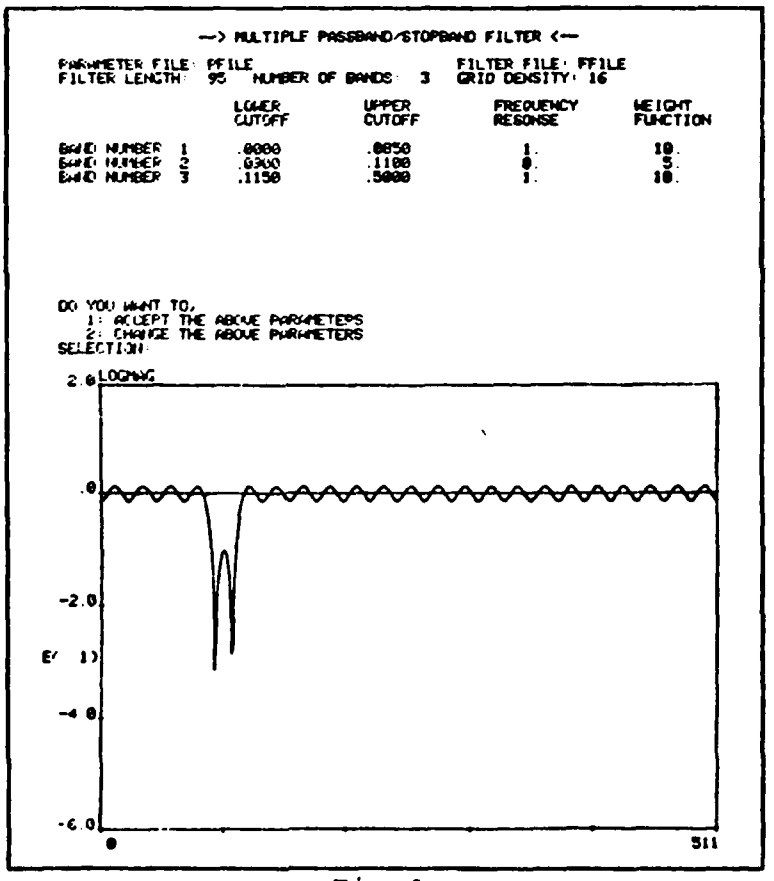


Fig 6

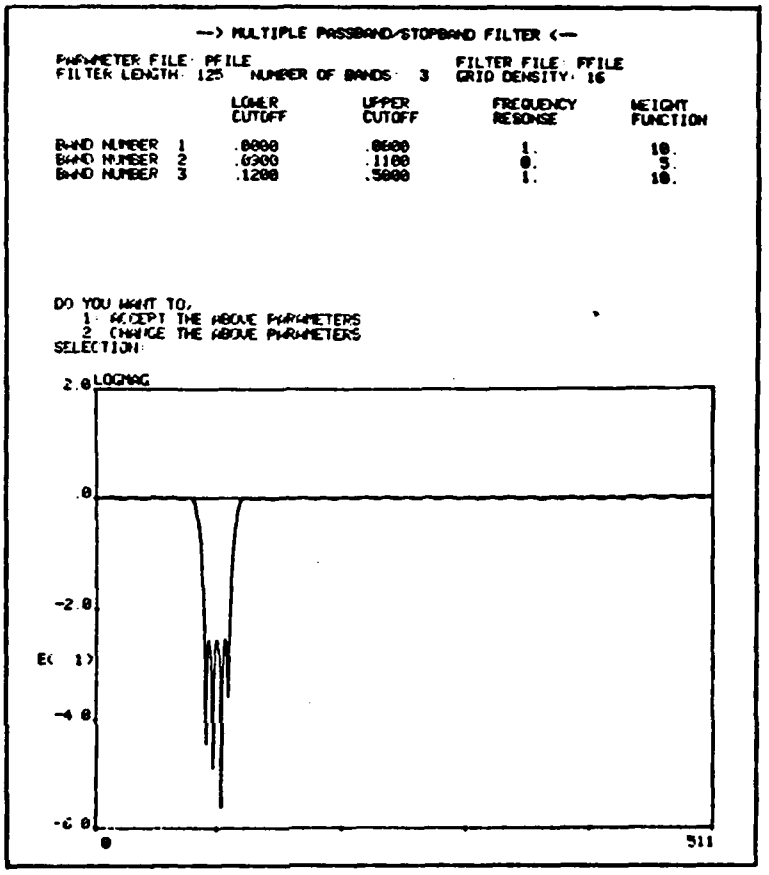

—> MULTIPLE PASSBAND/STOPBAND FILTER <—
PARAMETER FILE: PFILE
FILTER FILE: FFILE
FILTER LENGTH: 125
NUMBER OF BANDS: 3
GRID DENSITY: 16
LOWER CUTOFF
UPPER CUTOFF
FREQUENCY RESPONSE
WEIGHT FUNCTION
BAND NUMBER 1 .0000 .0600 1. 10.
BAND NUMBER 2 .0900 .1100 0. 5.
BAND NUMBER 3 .1200 .5000 1. 10.
DO YOU WANT TO,
1 ACCEPT THE ABOVE PARAMETERS
2 CHANGE THE ABOVE PARAMETERS
SELECTION:
LOGMAG
2.0
.0
-2.0
E(1)
-4.0
-6.0
0
511

Fig 7

The cutoff frequencies were again varied to reduce the length of the bandstop to only one ripple centered on the desired notch frequency. The final filter design is shown in Fig 8.

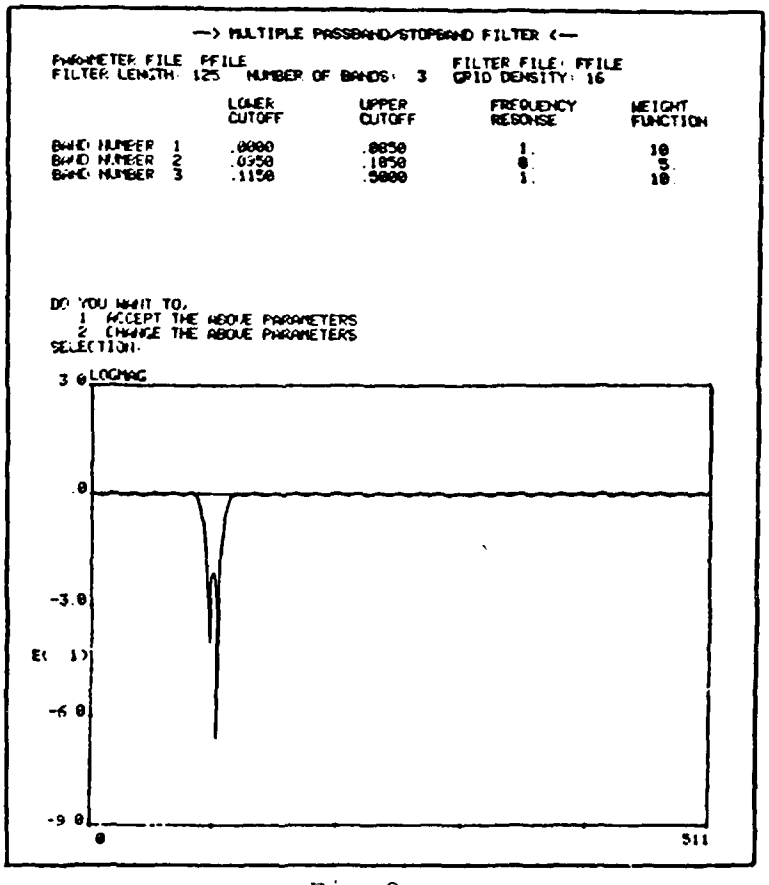


Fig 8

To illustrate the affect of even/odd filter length on the filter design, the length of the filter shown in Fig 8 was changed from 125 to 124 points. The result is shown in Fig 9.

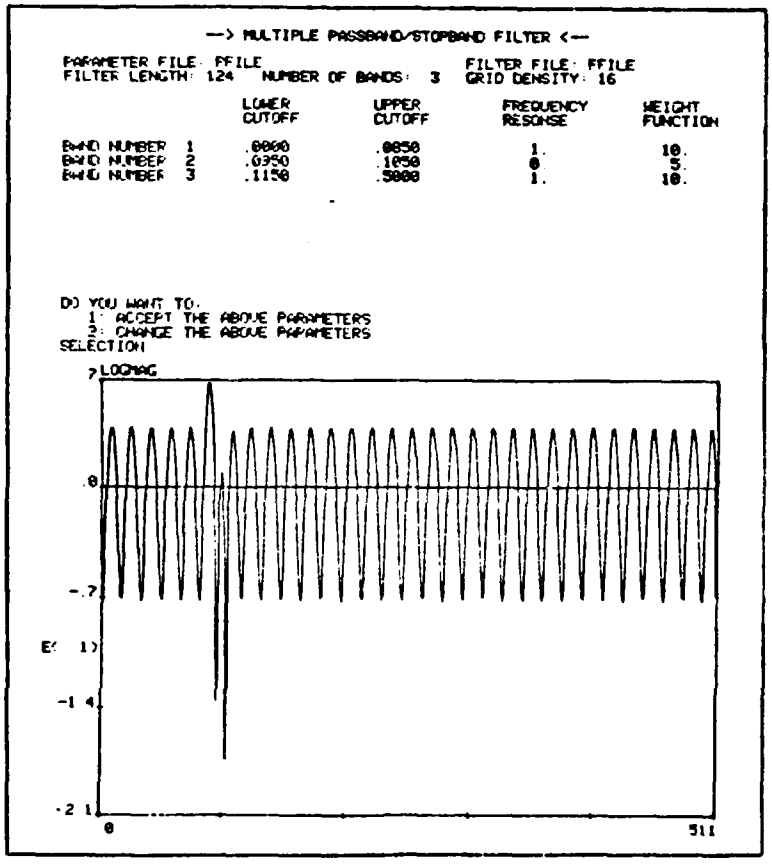
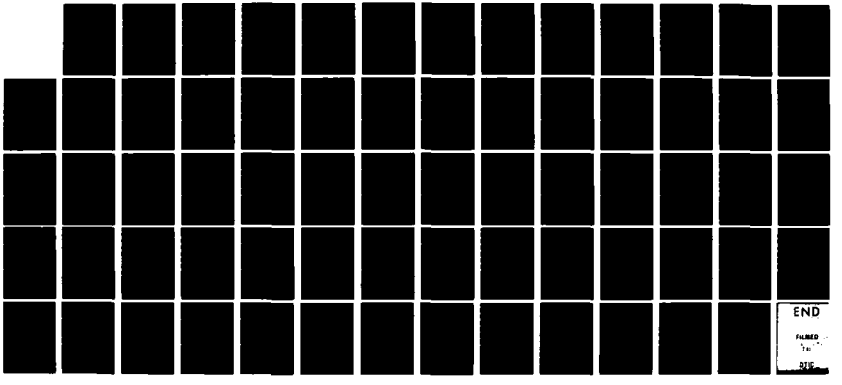


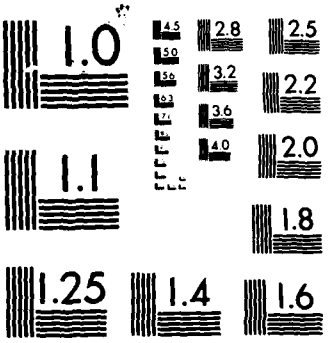
Fig 9

After the filter cutoff frequencies are adjusted as desired, the weight factor in each band can be adjusted to give the desired relative error.

AD-A124 750 EXPANSION OF THE ECLIPSE DIGITAL SIGNAL PROCESSING SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING G R ALLEN DEC 82 3/3

UNCLASSIFIED AFIT/GE/EE/82D-16 F/G 9/2 NL





```
C        file and is 88 bytes long. Both files are always deleted prior
C        to being created.

C        The filterscan have a maximum of 10 bands.

C*************************************************************************

      INTEGER PFILE(7),PF(2),SP,RET,FFILE(7),FF(2)
      LOGICAL SET,ITEST
C
C     Retrieve the command line files.
C
      CALL COMLN(RET,SP,PFILE,FFILE,SP,SP,PF,FF,SP)
      IF (RET.EQ.1) GO TO 45   ;if only one file
      IF (RET.EQ.2) GO TO 50   ;if two files
      GO TO 800

 45   SET=ITEST(PF(1),0)    ;check for P switch
      IF (SET) GO TO 55     ;if present, continue
      GO TO 800             ;if not, abort

 50   CALL SORT2(16,6,PFILE,FFILE,PF,FF) ;sort the command line files

 55   CALL CHOICE(PFILE,FFILE,PF,FF,RET)       ;obtain the parameter file

      IF (RET.EQ.1) GO TO 900      ;if no second file, then done
 60   CALL DESIGN(PFILE,FFILE,FF)  ;if second file present,then design filter
C
C     The D switch option is considered only if a filter was designed.
C
      CALL DELCHC(PFILE,PF)

      GO TO 900

 800  TYPE "(CR)
     *Incorrect command line. Consult program(CR)
     *documentation for the correct syntax.(CR)"

 900  CALL RESET
      CALL EXIT
      END

C*************************************************************************
```

```
C*************************************************************************

C    Title: LPFir
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program utilizes the Parks-McClellan algorithm to design
C        linear phase FIR filters. It can be used to design lowpass,
C        highpass, multiband, differentiators and Hilbert transform filters
C        with an impulse response between 3 and 256 points.

C     Environment:
C        This is a Fortran V program that has been designed to run on
C        a mapped-RDOS Eclipse S/250 minicomputer.  .

C     Compile command:
C        FORTRAN LPFIR

C     Load command:
C        RLDR/P LPFIR COMLN SORT2 FILCHC CHOICE STATUS RDBYTS SHOPAR^
C         NEWSCR DESIGN REMEZ WATE EFF D GEE OUCH @FLIB@

C     Command line:
C        LPFIR parameter/P [/E] [/D] [filter/F [/L] ]

C        where "parameter" and "filter" are any legal RDOS filename

C        The P switch must always be attached to the parameter filename. A
C        parameter file will be created with the filter parameters
C        interactively specified by the user. The filter parameters will
C        be displayed and can be changed if requested by the user.

C        The E switch denotes that the parameter file already exists. The
C        filter parameters will be display and can be changed if requested
C        by the user.

C        The filter filename and F switch denotes that the filter specified
C        by the parameter file will be designed and the impulse response
C        stored under the filter filename. The F switch must be attached.

C        The L switch denotes that a listing for the filter design will
C        be sent to the printer.

C        If the parameter and filter files are both given, they can be
C        typed in any order.

C        The D switch can only be attached to the parameter file if a
C        filter file is also specified. This switch deletes the parameter
C        file after the filter file has been created.

C     Comments:
C        The impulse response file will be created as a random file and will
C        contain real data. The parameter file is also created as a random
```

```
C*******************************************************************************

C     Title: Choice
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine is used by program LPFIR to collect filter design
C        parameters from the user.

C     Compile command:
C        FORTRAN CHOICE

C     Load command:
C        RLDR/P main program CHOICE STATUS RDBYTS SHOPAR NEWSCR etc

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        PFILE/PF              the filename that the filter design parameters
C                              will be written to and switch array

C        FFILE/FF              the filename that will contain the filter
C                              impulse (if one was requested by the user)
C                              and switch array

C        RET                   this integer variable is sent to the routine
C                              set to 1 (if FFILE does not exist) or 2
C                              (if the FFILE does exists)

C        The filenames and switch arrays are of the type returned by the
C        COMARG routine.

C*******************************************************************************

      SUBROUTINE CHOICE(PFILE,FFILE,PF,FF,RET)

      INTEGER PFILE(7),FFILE(7),PF(2),FF(2),YES,NO,KEEP,EXIST,BYTS
      INTEGER BLKS,LASTBYT,RET
      REAL PARA(44)

      LOGICAL ITEST,SET

      NO=0
      YES=1

      SET=ITEST(PF(1),11)       ;check for E switch
      IF (.NOT.SET) GO TO 10    ;if not present, collect parameters
C
C     This section of code collects the filter parameters from a disk
C     file.
C
      CALL STATUS(PFILE,BLKS,LASTBYT)
```

```
      BYTS=(BLKS*512)+LASTBYT
      CALL RDBYTS(PFILE,BYTS,PARA,88)
      EXIST=YES
      GO TO 70
C
C     This section of code collects the filter parameters interactively
C     from the user.
C
 10   EXIST=NO
      CALL NEWSCR

      ACCEPT "(CR)
     *Enter the filter length,(CR)
     *   (3-256): ",PARA(1)
      IF (PARA(1).GE.3. .AND. PARA(1).LE.256.) GO TO 20
      WRITE (10,1)
 1    FORMAT ("(CR)(CR)(CR)
     *Please make selections only from the given options.")
      GO TO 10

 20   ACCEPT "(CR)
     *Enter the type of filter,(CR)
     *   1: multiple passband/stopband filter(CR)
     *   2: differentiator(CR)
     *   3: Hilbert Transform filter(CR)
     *selection:",PARA(2)
      IF (PARA(2).GE.1. .AND. PARA(2).LE.3.) GO TO 30
      WRITE(10,1)
      GO TO 20

 30   ACCEPT "(CR)
     *Enter the number of bands,(CR)
     *   (1-10):",PARA(3)
      IF (PARA(3).GE.1. .AND. PARA(3).LE.10.) GO TO 40
      WRITE (10,1)
      GO TO 30

 40   ACCEPT "(CR)
     *Enter the grid density,(CR)
     *   (1-16):",PARA(4)
      IF (PARA(4).GE.1. .AND. PARA(4).LE.16.) GO TO 50
      WRITE(10,1)
      GO TO 40

 50   TYPE "(CR)(CR)(CR)(CR)(CR)(CR)(CR)
     *The following must be specified for each band,(CR)
     *(CR)
     *   lower cutoff freq(CR)
     *   upper cutoff freq"

      IF (PARA(2).EQ.1 .OR. PARA(2).EQ.3) TYPE "
     *   freq response"

      IF (PARA(2).EQ.2) TYPE "
```

```
*   slope"

 TYPE "
*   weight function"

 TYPE "(CR)
*where,(CR)
*(CR)
*   the lower and upper cutoff frequency specified(CR)
*   must be in the interval 1-.5(CR)
*   (this implies a sampling frequency of 1)(CR)"

 IF (PARA(2).EQ.1 .OR. PARA(2).EQ.3) TYPE "
*   the frequency response must be zero or(CR)
*   a positive value"

 IF (PARA(2).EQ.1 .OR. PARA(2).EQ.3) TYPE "(CR)
*   the weight function must be a(CR)
*   positive value"

 IF (PARA(2).EQ.2) TYPE "
*   the slope and weight function must(CR)
*   be a positive value"

 TYPE "(CR)
*Press carriage return to begin"
 ACCEPT

 I=0
 J=4
60   J=J+1
 I=I+1
 WRITE(10,6) I
6    FORMAT ("(CR)Band number ",I2)
 ACCEPT "
*lower cutoff freq : ",PARA(J)
 J=J+1
 ACCEPT "
*upper cutoff freq : ",PARA(J)
 J=J+1
 IF (PARA(2).EQ.1 .OR. PARA(2).EQ.3) ACCEPT "
*freq response: ",PARA(J)
 IF (PARA(2).EQ.2) ACCEPT "
*slope: ",PARA(J)
 J=J+1
 ACCEPT "
*weight function: ",PARA(J)
 IF (FLOAT(I).LT.PARA(3)) GO TO 60
C
C     Display the parameters and have the user decide if they will be kept.
C
70   CALL SHOPAR(PARA,PFILE,FFILE,RET,KEEP)
 IF (KEEP.EQ.NO) GO TO 10
 IF (KEEP.EQ.YES .AND. EXIST.EQ.NO) GO TO 75
```

```
      IF (KEEP.EQ.YES .AND. EXIST.EQ.YES) GO TO 90
      CALL ERROR("invalid value returned for KEEP")
C
C     This section of code writes the filter parameters to file, if the
C     parameter file does not already exist.
C
 75   CALL DFILW(PFILE,IER)
      CALL CFILW(PFILE,2,IER)
      IF (IER.NE.1) TYPE "CFILW error ",IER

      CALL OPEN(1,PFILE,3,IER)
      IF (IER.NE.1) TYPE "OPEN error ",IER

      BYTS=16+(PARA(3)*16)
      CALL WRSEQ(1,PARA,BYTS,IER)
      IF (IER.NE.1) TYPE "WRSEQ error ",IER

      CLOSE 1

 90   RETURN
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: ShoPar
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine is used by program LPFIR to display filter design
C        parameters. It also requests from the user whether the parameters
C        will be kept or changed and returns the decision to the calling
C        program.

C     Compile command:
C        FORTRAN SHOPAR

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        PARA          a 44-element array that contains the design parameters

C        PFILE         the name of the parameter file (in S format) that
C                      contains array PARA

C        FFILE         the name of the filter impulse response file (in S
C                      format) if this file was requested by the user

C        RET           this integer variable is sent to the routine set to
C                      1 (if FFILE was not requested) or 2 (if FFILE was
C                      requested)

C        KEEP          this integer variable is returned by the routine
C                      set to 1 (if the user decided to keep the design
C                      parameters) or 2 (if the user wants to change the
C                      design parameters)

C*******************************************************************************

      SUBROUTINE SHOPAR(PARA,PFILE,FFILE,RET,KEEP)

      INTEGER PFILE(7),FFILE(7),RET,KEEP,YES,NO
      REAL PARA(44)

      NO=0
      YES=1

      NFILT=INT(PARA(1))
      NBANDS=INT(PARA(3))
      LGRID=INT(PARA(4))

 10   TYPE
      IF (PARA(2).EQ.1.) TYPE "
     *                --) Multiple Passband/Stopband Filter (--"
      IF (PARA(2).EQ.2.) TYPE "
```

```
      #                          -->Differentiator <--"
       IF (PARA(2).EQ.3.) TYPE "
      #                   --> Hilbert Transform Filter <--"
       TYPE

       IF (RET.EQ.2) WRITE(10,3) PFILE(1),FFILE(1)
3      FORMAT("
      #Parameter File: ",S13,"              Filter File: ",S13)

       IF (RET.EQ.1) WRITE(10,4) PFILE(1)
4      FORMAT("
      #Parameter File: ",S13,"              Filter File: not specified")
       WRITE(10,2) NFILT,NBANDS,LGRID
2      FORMAT("
      #Filter Length: ",I3,"   Number of Bands: ",I2,"   Grid Density: ",I2)

       IF (PARA(2).EQ.1 .OR. PARA(2).EQ.3) TYPE "(CR)
      #                    Lower          Upper          Frequency       Weight(CR)
      #                    Cutoff         Cutoff         Resonse         Function"

       IF (PARA(2).EQ.2) TYPE "
      #                    Lower          Upper          Slope           Weight(CR)
      #                    Cutoff         Cutoff                         Function"

       TYPE
50     I=1
       J=5
60     WRITE(10,6) I,PARA(J),PARA(J+1),PARA(J+2),PARA(J+3)
6      FORMAT("
      #Band Number ",I2,"     ",F5.4,"         ",F5.4,"           ",F3.0,
      #"            ",F3.0)
       I=I+1
       J=J+4
       IF (FLOAT(I).LE.PARA(3)) GO TO 60

       DO 65 K=I,11
       TYPE
65     CONTINUE

80     ACCEPT "
      #Do you want to,(CR)
      #   1: accept the above parameters(CR)
      #   2: change the above parameters(CR)
      #selection: ",IKE

       KEEP=2
       IF (IKE.EQ.1) KEEP=YES
       IF (IKE.EQ.2) KEEP=NO
       IF (KEEP.NE.2) GO TO 90
       WRITE(10,1)
1      FORMAT("(CR)(CR)(CR)
      #Please select only from the options given.(CR)(CR)")
       TYPE "
      #Press carriage return to continue"
```

```
      ACCEPT
      GO TO 10

 90   RETURN
      END

C*************************************************************************
```

```
C*************************************************************************

C     Title: Design
C     Authors: James H. McClellan
C              Thomas W. Parks
C              Lawrence R. Rabiner
C     Date: Dec 73

C     Revised by: Lt Allen
C     Revision date: Dec 82

C     Function:
C        This routine implements the Parks-McClellan algorithm to design
C        a variety of linear phase FIR filters. For a discussion of this
C        algorithm, refer to section 5.1 of the following publication.

C                Programs for Digital Signal Processing
C                New York, IEEE Press, 1979

C                IEEE Book Numbers: Clothbound: 0-87942-127-4
C                                   Paperbound: 0-87942-128-2

C        The above publication contains the source code for the algorithm
C        that was revised and used in this subroutine. It also contains
C        the source code for the subroutines listed in the load line. The
C        revisions allow the program to be used as a subroutine with the
C        filter design parameters read from disk file and the filter
C        impulse response written to disk file.

C        This routine was designed for use with program LPFIR. Program LPFIR
C        should be consulted for a description of the type of filters that
C        this routine can be used to design. Subroutine CHOICE creates the
C        parameter file that is intended for use with this routine.

C     Compile command:
C        FORTRAN DESIGN

C     Load command:
C        RLDR/P main program DESIGN ERROR EFF REMEZ D GEE OUCH WATE etc

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        PFILE           the parameter filename

C        FFILE/FF        the filter filename and corresponding switch array

C        both filenames should be in S format and the switch array is of
C        the type returned by the COMARG subroutine

C*************************************************************************

      SUBROUTINE DESIGN PFILE,FFILE,FF)
```

```
C*************************************************************************
C
C     This section of code is identical to the referenced program.
C
      COMMON PI2,AD,DEV,X,Y,GRID,DES,WT,ALPHA,IEXT,NFCNS,NGRID
      COMMON /OOPS/NITER,IOUT
      DIMENSION IEXT(130),AD(130),ALPHA(130),X(130),Y(130)
      DIMENSION H(130)
      DIMENSION DES(2080),GRID(2080),WT(2080)
      DIMENSION EDGE(20),FX(10),WTX(10),DEVIAT(10)
      DOUBLE PRECISION PI2,PI
      DOUBLE PRECISION AD,DEV,X,Y
      DOUBLE PRECISION GEE,D
      INTEGER BD1,BD2,BD3,BD4
      DATA BD1,BD2,BD3,BD4/1HB,1HA,1HN,1HD/
      IOUT=10    ;this is the I/O unit number for the console
      PI=4.0*DATAN(1.0D0)
      PI2=2.0D00*PI

C*************************************************************************
C
C     This section of code is a revision to the referenced program.
C
C     Revision code parameters
C
      REAL PARA(44),IMPULSE(128)
      INTEGER PFILE(7),FFILE(7),FF(2),BYTS,BLKS,LASTBYT
      LOGICAL ITEST,SET
C
C     Get filter parameters from file.
C
      CALL STATUS(PFILE,BLKS,LASTBYT)
      BYTS=(BLKS*512)+LASTBYT
      CALL RDBYTS(PFILE,BYTS,PARA,44)
C
C     Align parameters with variables in the Parks-McClellan algorithm.
C
      NFILT=PARA(1)
      JTYPE=PARA(2)
      NBANDS=PARA(3)
      LGRID=PARA(4)

      L=0
      I=0
      J=4
 60   I=I+1
      J=J+1
      L=L+1
      EDGE(L)=PARA(J)
      J=J+1
      L=L+1
      EDGE(L)=PARA(J)
      J=J+1
```

```
      FX(I)=PARA(J)
      J=J+1
      WTX(I)=PARA(J)
      IF (I.LT.INT(PARA(3))) GO TO 60
C*******************************************************************************
C
C     This section of code is identical to the referenced program.
C
      NEG=1
      IF (JTYPE.EQ.1) NEG=0
      NODD=NFILT/2
      NODD=NFILT-2*NODD
      NFCNS=NFILT/2
      IF(NODD.EQ.1.AND.NEG.EQ.0) NFCNS=NFCNS+1
C
C SET UP THE DENSE GRID. THE NUMBER OF POINTS IN THE GRID
C IS (FILTER LENGTH + 1)*GRID DENSITY/2
C
      GRID(1)=EDGE(1)
      DELF=LGRID*NFCNS
      DELF=0.5/DELF
      IF (NEG.EQ.0) GO TO 135
      IF(EDGE(1).LT.DELF) GRID(1)=DELF
 135  CONTINUE
      J=1
      L=1
      LBAND=1
 140  FUP=EDGE(L+1)
 145  TEMP=GRID(J)
C
C CALCULATE THE DESIRED MAGNITUDE RESPONSE AND THE WEIGHT
C FUNCTION ON THE GRID
C
      DES(J)=EFF(TEMP,FX,WTX,LBAND,JTYPE)
      WT(J)=WATE(TEMP,FX,WTX,LBAND,JTYPE)
      J=J+1
      GRID(J)=TEMP+DELF
      IF(GRID(J).GT.FUP) GO TO 150
      GO TO 145
 150  GRID(J-1)=FUP
      DES(J-1)=EFF(FUP,FX,WTX,LBAND,JTYPE)
      WT(J-1)=WATE(FUP,FX,WTX,LBAND,JTYPE)
      LBAND=LBAND+1
      L=L+2
      IF(LBAND.GT.NBANDS) GO TO 160
      GRID(J)=EDGE(L)
      GO TO 140
 160  NGRID=J-1
      IF(NEG.NE.NODD) GO TO 165
      IF(GRID(NGRID).GT.(0.5-DELF)) NGRID=NGRID-1
 165  CONTINUE
C
C  SET UP A NEW APPROXIMATION PROBLEM WHICH IS EQUIVALENT
```

```
C  TO THE ORIGINAL PROBLEM
C
      IF(NEG) 170,170,180
  170 IF(NODD.EQ.1) GO TO 200
      DO 175 J=1,NGRID
      CHANGE=DCOS(PI*GRID(J))
      DES(J)=DES(J)/CHANGE
  175 WT(J)=WT(J)*CHANGE
      GO TO 200
  180 IF(NODD.EQ.1) GO TO 190
      DO 185 J=1,NGRID
      CHANGE=DSIN(PI*GRID(J))
      DES(J)=DES(J)/CHANGE
  185 WT(J)=WT(J)*CHANGE
      GO TO 200
  190 DO 195 J=1,NGRID
      CHANGE=DSIN(PI2*GRID(J))
      DES(J)=DES(J)/CHANGE
  195 WT(J)=WT(J)*CHANGE
C
C  INITIAL GUESS FOR THE EXTREMAL FREQUENCIES--EQUALLY
C  SPACED ALONG THE GRID
C
  200 TEMP=FLOAT(NGRID-1)/FLOAT(NFCNS)
      DO 210 J=1,NFCNS
      XT=J-1
  210 IEXT(J)=XT*TEMP+1.0
      IEXT(NFCNS+1)=NGRID
      NM1=NFCNS-1
      NZ=NFCNS+1
C
C  CALL THE REMEZ EXCHANGE ALGORITHM TO DO THE APPROXIMATION
C  PROBLEM
C
      CALL REMEZ
C
C  CALCULATE THE IMPULSE RESPONSE.
C
      IF(NEG) 300,300,320
  300 IF(NODD.EQ.0) GO TO 310
      DO 305 J=1,NM1
      NZMJ=NZ-J
  305 H(J)=0.5*ALPHA(NZMJ)
      H(NFCNS)=ALPHA(1)
      GO TO 350
  310 H(1)=0.25*ALPHA(NFCNS)
      DO 315 J=2,NM1
      NZMJ=NZ-J
      NF2J=NFCNS+2-J
  315 H(J)=0.25*(ALPHA(NZMJ)+ALPHA(NF2J))
      H(NFCNS)=0.5*ALPHA(1)+0.25*ALPHA(2)
      GO TO 350
  320 IF(NODD.EQ.0) GO TO 330
      H(1)=0.25*ALPHA(NFCNS)
```

```
      H(2)=0.25*ALPHA(NM1)
      DO 325 J=3,NM1
      NZMJ=NZ-J
      NF3J=NFCNS+3-J
  325 H(J)=0.25*(ALPHA(NZMJ)-ALPHA(NF3J))
      H(NFCNS)=0.5*ALPHA(1)-0.25*ALPHA(3)
      H(NZ)=0.0
      GO TO 350
  330 H(1)=0.25*ALPHA(NFCNS)
      DO 335 J=2,NM1
      NZMJ=NZ-J
      NF2J=NFCNS+2-J
  335 H(J)=0.25*(ALPHA(NZMJ)-ALPHA(NF2J))
      H(NFCNS)=0.5*ALPHA(1)-0.25*ALPHA(2)

C*************************************************************************
C
C     This section of code is a revision to the referenced program.
C
C     Calculate the complete impulse response.
C
  350 CONTINUE
      DO 351 J=1,NFCNS
      K=NFILT+1-J
      IF (NEG.EQ.0) IMPULSE(J)=H(J)
      IF (NEG.EQ.0) IMPULSE(K)=H(J)
      IF (NEG.EQ.1) IMPULSE(J)=H(J)
      IF (NEG.EQ.1) IMPULSE(K)=-H(J)
  351 CONTINUE
      IF (NEG.EQ.1 .AND. NODD.EQ.1) IMPULSE(NZ)=0.0
C
C     Write filter impulse response to file.
C
  352 CALL DFILW(FFILE,IER)
      IF (IER.EQ.13) GO TO 353
      IF (IER.NE.1) TYPE "DFILW error ",IER," with filter file"
  353 CALL CFILW(FFILE,2,IER)
      IF (IER.NE.1) TYPE "CFILW error ",IER," with filter file"
      CALL OPEN(2,FFILE,3,IER)
      IF (IER.NE.1) TYPE "OPEN error ",IER," with filter file"

      BYTS=NFILT*4
      CALL WRSEQ(2,IMPULSE,BYTS,IER)
      IF (IER.NE.1) TYPE "WRSEQ error ",IER," with filter file"

      CALL CLOSE(2,IER)
      IF (IER.NE.1) TYPE "CLOSE error ",IER," with filter file"

      SET=ITEST(FF(1),4)   ;true if L switch is present
      IF (.NOT.SET) GO TO 500  ;if L switch is not present
                               ;then do not send a filter design
                               ;listing to the printer

C*************************************************************************
```

```
C
C     This section of code is identical to the referenced program.
C
C  PROGRAM OUTPUT SECTION.
C
      WRITE(IOUT,360)
  360 FORMAT(1H1, 70(1H*)//15X,29HFINITE IMPULSE RESPONSE (FIR)/
     113X,34HLINEAR PHASE DIGITAL FILTER DESIGN/
     217X,24HREMEZ EXCHANGE ALGORITHM/)
      IF(JTYPE.EQ.1) WRITE(IOUT,365)
  365 FORMAT(22X,15HBANDPASS FILTER/)
      IF(JTYPE.EQ.2) WRITE(IOUT,370)
  370 FORMAT(22X,14HDIFFERENTIATOR/)
      IF(JTYPE.EQ.3) WRITE(IOUT,375)
  375 FORMAT(20X,19HHILBERT TRANSFORMER/)
      WRITE(IOUT,378) NFILT
  378 FORMAT(20X,16HFILTER LENGTH = ,I3/)
      WRITE(IOUT,380)
  380 FORMAT(15X,28H***** IMPULSE RESPONSE *****)
      DO 381 J=1,NFCNS
      K=NFILT+1-J
      IF(NEG.EQ.0) WRITE(IOUT,382) J,H(J),K
      IF(NEG.EQ.1) WRITE(IOUT,383) J,H(J),K
  381 CONTINUE
  382 FORMAT(13X,2HH(,I2,4H) = ,E15.8,5H = H(,I3,1H))
  383 FORMAT(13X,2HH(,I2,4H) = ,E15.8,6H = -H(,I3,1H))
      IF(NEG.EQ.1.AND.NODD.EQ.1) WRITE(IOUT,384) NZ
  384 FORMAT(13X,2HH(,I2,8H) =  0.0)
      DO 450 K=1,NBANDS,4
      KUP=K+3
      IF(KUP.GT.NBANDS) KUP=NBANDS
      WRITE(IOUT,385) (BD1,BD2,BD3,BD4,J,J=K,KUP)
  385 FORMAT(/24X,4(4A1,I3,7X))
      WRITE(IOUT,390) (EDGE(2*J-1),J=K,KUP)
  390 FORMAT(2X,15HLOWER BAND EDGE,5F14.7)
      WRITE(IOUT,395) (EDGE(2*J),J=K,KUP)
  395 FORMAT(2X,15HUPPER BAND EDGE,5F14.7)
      IF(JTYPE.NE.2) WRITE(IOUT,400) (FX(J),J=K,KUP)
  400 FORMAT(2X,13HDESIRED VALUE,2X,5F14.7)
      IF(JTYPE.EQ.2) WRITE(IOUT,405) (FX(J),J=K,KUP)
  405 FORMAT(2X,13HDESIRED SLOPE,2X,5F14.7)
      WRITE(IOUT,410) (WTX(J),J=K,KUP)
  410 FORMAT(2X,9HWEIGHTING,6X,5F14.7)
      DO 420 J=K,KUP
  420 DEVIAT(J)=DEV/WTX(J)
      WRITE(IOUT,425) (DEVIAT(J),J=K,KUP)
  425 FORMAT(2X,9HDEVIATION,6X,5F14.7)
      IF(JTYPE.NE.1) GO TO 450
      DO 430 J=K,KUP
  430 DEVIAT(J)=20.0*ALOG10(DEVIAT(J)+FX(J))
      WRITE(IOUT,435) (DEVIAT(J),J=K,KUP)
  435 FORMAT(2X,15HDEVIATION IN DB,5F14.7)
  450 CONTINUE
      DO 452 J=1,NZ
```

```
      IX=IEXT(J)
  452 GRID(J)=GRID(IX)
      WRITE(IOUT,455) (GRID(J),J=1,NZ)
  455 FORMAT(/2X,47HEXTREMAL FREQUENCIES--MAXIMA OF THE ERROR CURVE/
     1 (2X,5F12.7))
      WRITE(IOUT,460)
  460 FORMAT(/1X,70(1H*)/1H1)

C*******************************************************************************

  500 RETURN
      END

C*******************************************************************************
```

```
C
C-----------------------------------------------------------------------
C SUBROUTINE: REMEZ
C   THIS SUBROUTINE IMPLEMENTS THE REMEZ EXCHANGE ALGORITHM
C   FOR THE WEIGHTED CHEBYSHEV APPROXIMATION OF A CONTINUOUS
C   FUNCTION WITH A SUM OF COSINES.  INPUTS TO THE SUBROUTINE
C   ARE A DENSE GRID WHICH REPLACES THE FREQUENCY AXIS, THE
C   DESIRED FUNCTION ON THIS GRID, THE WEIGHT FUNCTION ON THE
C   GRID, THE NUMBER OF COSINES, AND AN INITIAL GUESS OF THE
C   EXTREMAL FREQUENCIES.  THE PROGRAM MINIMIZES THE CHEBYSHEV
C   ERROR BY DETERMINING THE BEST LOCATION OF THE EXTREMAL
C   FREQUENCIES (POINTS OF MAXIMUM ERROR) AND THEN CALCULATES
C   THE COEFFICIENTS OF THE BEST APPROXIMATION.
C-----------------------------------------------------------------------
C
      SUBROUTINE REMEZ
      COMMON PI2,AD,DEV,X,Y,GRID,DES,WT,ALPHA,IEXT,NFCNS,NGRID
      COMMON /OOPS/NITER,IOUT
      DIMENSION IEXT(130),AD(130),ALPHA(130),X(130),Y(130)
      DIMENSION DES(2080),GRID(2080),WT(2080)
      DIMENSION A(66),P(65),Q(65)
      DOUBLE PRECISION PI2,DNUM,DDEN,DTEMP,A,P,Q
      DOUBLE PRECISION DK,DAK
      DOUBLE PRECISION AD,DEV,X,Y
      DOUBLE PRECISION GEE,D
C
C  THE PROGRAM ALLOWS A MAXIMUM NUMBER OF ITERATIONS OF 25
C
      ITRMAX=25
      DEVL=-1.0
      NZ=NFCNS+1
      NZZ=NFCNS+2
      NITER=0
  100 CONTINUE
      IEXT(NZZ)=NGRID+1
      NITER=NITER+1
      IF(NITER.GT.ITRMAX) GO TO 400
      DO 110 J=1,NZ
      JXT=IEXT(J)
      DTEMP=GRID(JXT)
      DTEMP=DCOS(DTEMP*PI2)
  110 X(J)=DTEMP
      JET=(NFCNS-1)/15+1
      DO 120 J=1,NZ
  120 AD(J)=D(J,NZ,JET)
      DNUM=0.0
      DDEN=0.0
      K=1
      DO 130 J=1,NZ
      L=IEXT(J)
      DTEMP=AD(J)*DES(L)
      DNUM=DNUM+DTEMP
      DTEMP=FLOAT(K)*AD(J)/WT(L)
      DDEN=DDEN+DTEMP
```

```
  130 K=-K
      DEV=DNUM/DDEN
      WRITE(IOUT,131) DEV
  131 FORMAT(1X,12HDEVIATION = ,F12.9)
      NU=1
      IF(DEV.GT.0.0) NU=-1
      DEV=-FLOAT(NU)*DEV
      K=NU
      DO 140 J=1,NZ
      L=IEXT(J)
      DTEMP=FLOAT(K)*DEV/WT(L)
      Y(J)=DES(L)+DTEMP
  140 K=-K
      IF(DEV.GT.DEVL) GO TO 150
      CALL OUCH
      GO TO 400
  150 DEVL=DEV
      JCHNGE=0
      K1=IEXT(1)
      KNZ=IEXT(NZ)
      KLOW=0
      NUT=-NU
      J=1
C
C  SEARCH FOR THE EXTREMAL FREQUENCIES OF THE BEST
C  APPROXIMATION
C
  200 IF(J.EQ.NZZ) YNZ=COMP
      IF(J.GE.NZZ) GO TO 300
      KUP=IEXT(J+1)
      L=IEXT(J)+1
      NUT=-NUT
      IF(J.EQ.2) Y1=COMP
      COMP=DEV
      IF(L.GE.KUP) GO TO 220
      ERR=GEE(L,NZ)
      ERR=(ERR-DES(L))*WT(L)
      DTEMP=FLOAT(NUT)*ERR-COMP
      IF(DTEMP.LE.0.0) GO TO 220
      COMP=FLOAT(NUT)*ERR
  210 L=L+1
      IF(L.GE.KUP) GO TO 215
      ERR=GEE(L,NZ)
      ERR=(ERR-DES(L))*WT(L)
      DTEMP=FLOAT(NUT)*ERR-COMP
      IF(DTEMP.LE.0.0) GO TO 215
      COMP=FLOAT(NUT)*ERR
      GO TO 210
  215 IEXT(J)=L-1
      J=J+1
      KLOW=L-1
      JCHNGE=JCHNGE+1
      GO TO 200
  220 L=L-1
```

```
225 L=L-1
    IF(L.LE.KLOW) GO TO 250
    ERR=GEE(L,NZ)
    ERR=(ERR-DES(L))*WT(L)
    DTEMP=FLOAT(NUT)*ERR-COMP
    IF(DTEMP.GT.0.0) GO TO 230
    IF(JCHNGE.LE.0) GO TO 225
    GO TO 260
230 COMP=FLOAT(NUT)*ERR
235 L=L-1
    IF(L.LE.KLOW) GO TO 240
    ERR=GEE(L,NZ)
    ERR=(ERR-DES(L))*WT(L)
    DTEMP=FLOAT(NUT)*ERR-COMP
    IF(DTEMP.LE.0.0) GO TO 240
    COMP=FLOAT(NUT)*ERR
    GO TO 235
240 KLOW=IEXT(J)
    IEXT(J)=L+1
    J=J+1
    JCHNGE=JCHNGE+1
    GO TO 200
250 L=IEXT(J)+1
    IF(JCHNGE.GT.0) GO TO 215
255 L=L+1
    IF(L.GE.KUP) GO TO 260
    ERR=GEE(L,NZ)
    ERR=(ERR-DES(L))*WT(L)
    DTEMP=FLOAT(NUT)*ERR-COMP
    IF(DTEMP.LE.0.0) GO TO 255
    COMP=FLOAT(NUT)*ERR
    GO TO 210
260 KLOW=IEXT(J)
    J=J+1
    GO TO 200
300 IF(J.GT.NZZ) GO TO 320
    IF(K1.GT.IEXT(1)) K1=IEXT(1)
    IF(KNZ.LT.IEXT(NZ)) KNZ=IEXT(NZ)
    NUT1=NUT
    NUT=-NU
    L=0
    KUP=K1
    COMP=YNZ*(1.00001)
    LUCK=1
310 L=L+1
    IF(L.GE.KUP) GO TO 315
    ERR=GEE(L,NZ)
    ERR=(ERR-DES(L))*WT(L)
    DTEMP=FLOAT(NUT)*ERR-COMP
    IF(DTEMP.LE.0.0) GO TO 310
    COMP=FLOAT(NUT)*ERR
    J=NZZ
    GO TO 210
315 LUCK=6
```

```
      GO TO 325
  320 IF(LUCK.GT.9) GO TO 350
      IF(COMP.GT.Y1) Y1=COMP
      K1=IEXT(NZZ)
  325 L=NGRID+1
      KLOW=KNZ
      NUT=-NUT1
      COMP=Y1*(1.00001)
  330 L=L-1
      IF(L.LE.KLOW) GO TO 340
      ERR=GEE(L,NZ)
      ERR=(ERR-DES(L))*WT(L)
      DTEMP=FLOAT(NUT)*ERR-COMP
      IF(DTEMP.LE.0.0) GO TO 330
      J=NZZ
      COMP=FLOAT(NUT)*ERR
      LUCK=LUCK+10
      GO TO 235
  340 IF(LUCK.EQ.6) GO TO 370
      DO 345 J=1,NFCNS
      NZZMJ=NZZ-J
      NZMJ=NZ-J
  345 IEXT(NZZMJ)=IEXT(NZMJ)
      IEXT(1)=K1
      GO TO 100
  350 KN=IEXT(NZZ)
      DO 360 J=1,NFCNS
  360 IEXT(J)=IEXT(J+1)
      IEXT(NZ)=KN
      GO TO 100
  370 IF(JCHNGE.GT.0) GO TO 100
C
C  CALCULATION OF THE COEFFICIENTS OF THE BEST APPROXIMATION
C  USING THE INVERSE DISCRETE FOURIER TRANSFORM
C
  400 CONTINUE
      NM1=NFCNS-1
      FSH=1.0E-06
      GTEMP=GRID(1)
      X(NZZ)=-2.0
      CN=2*NFCNS-1
      DELF=1.0/CN
      L=1
      KKK=0
      IF(GRID(1).LT.0.01.AND.GRID(NGRID).GT.0.49) KKK=1
      IF(NFCNS.LE.3) KKK=1
      IF(KKK.EQ.1) GO TO 405
      DTEMP=DCOS(PI2*GRID(1))
      DNUM=DCOS(PI2*GRID(NGRID))
      AA=2.0/(DTEMP-DNUM)
      BB=-(DTEMP+DNUM)/(DTEMP-DNUM)
  405 CONTINUE
      DO 430 J=1,NFCNS
      FT=J-1
```

```
      FT=FT*DELF
      XT=DCOS(PI2*FT)
      IF(KKK.EQ.1) GO TO 410
      XT=(XT-BB)/AA
      XT1=SQRT(1.0-XT*XT)
      FT=ATAN2(XT1,XT)/PI2
  410 XE=X(L)
      IF(XT.GT.XE) GO TO 420
      IF((XE-XT).LT.FSH) GO TO 415
      L=L+1
      GO TO 410
  415 A(J)=Y(L)
      GO TO 425
  420 IF((XT-XE).LT.FSH) GO TO 415
      GRID(1)=FT
      A(J)=GEE(1,NZ)
  425 CONTINUE
      IF(L.GT.1) L=L-1
  430 CONTINUE
      GRID(1)=GTEMP
      DDEN=PI2/CN
      DO 510 J=1,NFCNS
      DTEMP=0.0
      DNUM=J-1
      DNUM=DNUM*DDEN
      IF(NM1.LT.1) GO TO 505
      DO 500 K=1,NM1
      DAK=A(K+1)
      DK=K
  500 DTEMP=DTEMP+DAK*DCOS(DNUM*DK)
  505 DTEMP=2.0*DTEMP+A(1)
  510 ALPHA(J)=DTEMP
      DO 550 J=2,NFCNS
  550 ALPHA(J)=2.0*ALPHA(J)/CN
      ALPHA(1)=ALPHA(1)/CN
      IF(KKK.EQ.1) GO TO 545
      P(1)=2.0*ALPHA(NFCNS)*BB+ALPHA(NM1)
      P(2)=2.0*AA*ALPHA(NFCNS)
      Q(1)=ALPHA(NFCNS-2)-ALPHA(NFCNS)
      DO 540 J=2,NM1
      IF(J.LT.NM1) GO TO 515
      AA=0.5*AA
      BB=0.5*BB
  515 CONTINUE
      P(J+1)=0.0
      DO 520 K=1,J
      A(K)=P(K)
  520 P(K)=2.0*BB*A(K)
      P(2)=P(2)+A(1)*2.0*AA
      JM1=J-1
      DO 525 K=1,JM1
  525 P(K)=P(K)+Q(K)+AA*A(K+1)
      JP1=J+1
      DO 530 K=3,JP1
```

```
530 P(K)=P(K)+AA*A(K-1)
    IF(J.EQ.NM1) GO TO 540
    DO 535 K=1,J
535 Q(K)=-A(K)
    NF1J=NFCNS-1-J
    Q(1)=Q(1)+ALPHA(NF1J)
540 CONTINUE
    DO 543 J=1,NFCNS
543 ALPHA(J)=P(J)
545 CONTINUE
    IF(NFCNS.GT.3) RETURN
    ALPHA(NFCNS+1)=0.0
    ALPHA(NFCNS+2)=0.0
    RETURN
    END
```

```
C
C-----------------------------------------------------------------------
C FUNCTION: WATE
C   FUNCTION TO CALCULATE THE WEIGHT FUNCTION AS A FUNCTION
C   OF FREQUENCY.  SIMILAR TO THE FUNCTION EFF, THIS FUNCTION CAN
C   BE REPLACED BY A USER-WRITTEN ROUTINE TO CALCULATE ANY
C   DESIRED WEIGHTING FUNCTION.
C-----------------------------------------------------------------------
C
      FUNCTION WATE(FREQ,FX,WTX,LBAND,JTYPE)
      DIMENSION FX(5),WTX(5)
      IF(JTYPE.EQ.2) GO TO 1
      WATE=WTX(LBAND)
      RETURN
    1 IF(FX(LBAND).LT.0.0001) GO TO 2
      WATE=WTX(LBAND)/FREQ
      RETURN
    2 WATE=WTX(LBAND)
      RETURN
      END
```

```
C
C-----------------------------------------------------------------------
C FUNCTION: EFF
C   FUNCTION TO CALCULATE THE DESIRED MAGNITUDE RESPONSE
C   AS A FUNCTION OF FREQUENCY.
C   AN ARBITRARY FUNCTION OF FREQUENCY CAN BE
C   APPROXIMATED IF THE USER REPLACES THIS FUNCTION
C   WITH THE APPROPRIATE CODE TO EVALUATE THE IDEAL
C   MAGNITUDE.  NOTE THAT THE PARAMETER FREQ IS THE
C   VALUE OF NORMALIZED FREQUENCY NEEDED FOR EVALUATION.
C-----------------------------------------------------------------------
C
      FUNCTION EFF(FREQ,FX,WTX,LBAND,JTYPE)
      DIMENSION FX(5),WTX(5)
      IF(JTYPE.EQ.2) GO TO 1
      EFF=FX(LBAND)
      RETURN
    1 EFF=FX(LBAND)*FREQ
      RETURN
      END
```

```
C
C-----------------------------------------------------------------------
C FUNCTION: D
C   FUNCTION TO CALCULATE THE LAGRANGE INTERPOLATION
C   COEFFICIENTS FOR USE IN THE FUNCTION GEE.
C-----------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION D(K,N,M)
      COMMON PI2,AD,DEV,X,Y,GRID,DES,WT,ALPHA,IEXT,NFCNS,NGRID
      DIMENSION IEXT(130),AD(130),ALPHA(130),X(130),Y(130)
      DIMENSION DES(2080),GRID(2080),WT(2080)
      DOUBLE PRECISION AD,DEV,X,Y
      DOUBLE PRECISION Q
      DOUBLE PRECISION PI2
      D=1.0
      Q=X(K)
      DO 3 L=1,M
      DO 2 J=L,N,M
      IF(J-K)1,2,1
    1 D=2.0*D*(Q-X(J))
    2 CONTINUE
    3 CONTINUE
      D=1.0/D
      RETURN
      END
```

```
C
C-----------------------------------------------------------------------
C FUNCTION: GEE
C   FUNCTION TO EVALUATE THE FREQUENCY RESPONSE USING THE
C   LAGRANGE INTERPOLATION FORMULA IN THE BARYCENTRIC FORM
C-----------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION GEE(K,N)
      COMMON PI2,AD,DEV,X,Y,GRID,DES,WT,ALPHA,IEXT,NFCNS,NGRID
      DIMENSION IEXT(130),AD(130),ALPHA(130),X(130),Y(130)
      DIMENSION DES(2080),GRID(2080),WT(2080)
      DOUBLE PRECISION P,C,D,XF
      DOUBLE PRECISION PI2
      DOUBLE PRECISION AD,DEV,X,Y
      P=0.0
      XF=GRID(K)
      XF=DCOS(PI2*XF)
      D=0.0
      DO 1 J=1,N
      C=XF-X(J)
      C=AD(J)/C
      D=D+C
    1 P=P+C*Y(J)
      GEE=P/D
      RETURN
      END
```

```
C
C-----------------------------------------------------------------------
C SUBROUTINE: OUCH
C   WRITES AN ERROR MESSAGE WHEN THE ALGORITHM FAILS TO
C   CONVERGE.  THERE SEEM TO BE TWO CONDITIONS UNDER WHICH
C   THE ALGORITHM FAILS TO CONVERGE: (1) THE INITIAL
C   GUESS FOR THE EXTREMAL FREQUENCIES IS SO POOR THAT
C   THE EXCHANGE ITERATION CANNOT GET STARTED, OR
C   (2) NEAR THE TERMINATION OF A CORRECT DESIGN,
C   THE DEVIATION DECREASES DUE TO ROUNDING ERRORS
C   AND THE PROGRAM STOPS.  IN THIS LATTER CASE THE
C   FILTER DESIGN IS PROBABLY ACCEPTABLE, BUT SHOULD
C   BE CHECKED BY COMPUTING A FREQUENCY RESPONSE.
C-----------------------------------------------------------------------
C
      SUBROUTINE OUCH
      COMMON /OOPS/NITER,IOUT
      WRITE(IOUT,1)NITER
    1 FORMAT(44H ********** FAILURE TO CONVERGE **********/
     141HOPROBABLE CAUSE IS MACHINE ROUNDING ERROR/
     223HONUMBER OF ITERATIONS =,I4/
     339HOIF THE NUMBER OF ITERATIONS EXCEEDS 3,/
     462HOTHE DESIGN MAY BE CORRECT, BUT SHOULD BE VERIFIED WITH AN FFT)
      RETURN
      END
```

Appendix F

Source Code for Support Software

```
C*******************************************************************************

C     Title: ComLn
C     Author: Lt Allen
C     Date: Dec  82

C     Function:
C        This routine fetches the number of files, filenames and switch
C        values of up to 3 files that may have been entered in the
C        command line with the executing program.

C     Compile command:
C        FORTRAN COMLN

C     Command line:
C        CALL COMLN(TOTAL,MAIN,FILE1,FILE2,FILE3,MS,F1,F2,F3)

C        where,

C        TOTAL returns the number of files, in addition to the executing
C        program, that were entered in the command line of the executing
C        program.

C        MAIN/MS are the name and switch values entered for the executing
C        program.

C        FILE1/F1, FILE2/F2 and FILE3/F3 are the additional filenames
C        and corresponding switches if entered in the command line.

C        All filenames are returned in the S format. The following table
C        gives the bit that is set in the switch array for each switch
C        that is attached to a filename. If the switch is not attached,
C        then the corresponding bit will be zero. Also, the unused bits
C        in the second switch element will be returned zero. The bits
C        are numbered from 0, the rightmost, to 15, the leftmost. This is
C        the convention used by the ITEST subroutine.

C        switch   bit of SW(1)       switch    bit of SW(2)

C           A         15              Q         15
C           B         14              R         14
C           C         13              S         13
C           D         12              T         12
C           E         11              U         11
C           F         10              V         10
C           G          9              W          9
C           H          8              X          8
C           I          7              Y          7
C           J          6              Z          6
C           K          5
C           L          4
C           M          3
C           N          2
C           O          1
```

```
C           P           0

C*************************************************************************

      SUBROUTINE COMLN(TOTAL,MAIN,FILE1,FILE2,FILE3,MS,F1,F2,F3)

      INTEGER MAIN(7),FILE1(7),FILE2(7),FILE3(7)
      INTEGER MS(2),F1(2),F2(2),F3(2),TOTAL

      TOTAL=0
      CALL GROUND(I)
      IF (I.EQ.0) OPEN 1,"COM.CM"     ;operating on background terminal
      IF (I.EQ.1) OPEN 1,"FCOM.CM"    ;operating on foreground terminal

      CALL COMARG(1,MAIN,MS,IER)
      IF (IER.NE.1) TYPE "COMARG error",IER," with main file"
      CALL COMARG(1,FILE1,F1,IER)
      IF (IER.EQ.9) GO TO 10
      IF (IER.NE.1) TYPE "COMARG error",IER," with first file"
      TOTAL=TOTAL+1

      CALL COMARG(1,FILE2,F2,IER)
      IF (IER.EQ.9) GO TO 10
      IF (IER.NE.1) TYPE "COMARG error",IER," with second file"
      TOTAL=TOTAL+1

      CALL COMARG(1,FILE3,F3,IER)
      IF (IER.EQ.9) GO TO 10
      IF (IER.NE.1) TYPE "COMARG error",IER," with third file"
      TOTAL=TOTAL+1

 10   CLOSE 1
      RETURN
      END

C*************************************************************************
```

```
C*******************************************************************************

C     Title: ClkSet
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine allows the user to interactively set the clock to
C        be used for an Eclipse A/D/A conversion operation.

C     Compile command:
C        FORTRAN CLKSET

C     Comments:
C        The device number (21 for A/D or 23 for D/A) is sent to the
C        routine in variable DEVICE.

C        The clock chosen is returned to the calling program in variable
C        CLOCK.

C*******************************************************************************

      SUBROUTINE CLKSET(DEVICE,CLOCK)

      INTEGER DEVICE,CLOCK

C*******************************************************************************

      IF (DEVICE.EQ.21 .OR. DEVICE.EQ.23) GO TO 10
      CALL ERROR("improper device number")

 10   TYPE "(CR)
     #What type of clock?(CR)
     #   1: pulse(CR)
     #   2: external(CR)
     #   3: internal"
      IF (DEVICE.EQ.21) TYPE "   4: DCH"   ;not allowed for A/D operations
      ACCEPT "
     #selection:",ICLOCK

      CLOCK=777K
      IF (ICLOCK.EQ.1) CLOCK=0K
      IF (ICLOCK.EQ.2) CLOCK=60000K
      IF (ICLOCK.EQ.3) CLOCK=40000K
      IF ((DEVICE.EQ.21) .AND.(ICLOCK.EQ.4)) CLOCK=20000K
      IF (CLOCK.NE.777K) GO TO 15
      WRITE (10,1)
 1    FORMAT ("(CR)(CR)(CR)
     #Please make selections only from the given options.")
      GO TO 10

 15   IF (CLOCK.EQ.0K) TYPE "(7)(7)(7)(CR)
     #Use of the pulse clock requires special software setup(CR)
     #and should not be attempted without consulting the SAM(CR)
```

```
     #User's Manual."
      IMIS=1

16    IF (CLOCK.EQ.OK) ACCEPT "(CR)
     #Do you want to,(CR)
     #   1: use pulse clock(CR)
     #   2: select another clock(CR)
     #selection:",IMIS

      IF (IMIS.EQ.1) GO TO 20
      IF (IMIS.EQ.2) GO TO 10
      WRITE(10,1)
      GO TO 16

20    RETURN
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: DelChc
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine deletes a disk file if it's switch array has the D
C        D switch set. The switch array should be of the form returned
C        by the COMARG call.

C     Compile command:
C        FORTRAN DELCHC

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        FILENAM                 the disk filename (in S format)

C        FS                      the corresponding switch array for the
C                                disk file

C*******************************************************************************

      SUBROUTINE DELCHC(FILENAM,FS)

      INTEGER FILENAM(7),FS(2)
      LOGICAL ITEST

      IF (.NOT.ITEST(FS(1),12)) GO TO 10
      CALL DFILW(FILENAM,IER)
      IF (IER.NE.1) WRITE(10,1) IER,FILENAM(1)
 1    FORMAT("
     *DFILW error ",I2," with file ",S13)

 10   RETURN
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: FilChc
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine verifies that two filenames are not identical. If
C        they are, an error message is printed to the screen and the
C        program is aborted.

C     Compile command:
C        FORTRAN FILCHC

C     Comments:
C        The filenames sent to this routine should be in the S format.
C        (This is the format returned by the COMARG call)

C*******************************************************************************

      SUBROUTINE FILCHC(FILE1,FILE2)

      INTEGER FILE1(7),FILE2(7),TEST

      TEST=0
      I=1
 10   IF (FILE1(I).NE.FILE2(I)) GO TO 20
      I=I+1
      IF (I.LT.8) GO TO 10
      GO TO 30

 20   TEST=1

 30   IF (TEST.EQ.0)
     #CALL ERROR("the given command line filenames cannot be identical")

      RETURN
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: FiltPlot
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program plots filter responses on the tetronix terminal. It
C        will plot an impulse of up to 512 points or the magnitude or log
C        magnitude of the 1024-point DFT magnitude of the impulse
C        response. The program assumes that the filter file is of
C        the type (time or frequency magnitude) specified to be
C        plotted. Only the first half (512 points) of the DFT files
C        are plotted. All file data types should be real.

C     Compile command:
C        FORTRAN FILTPLOT

C     Load command:
C        RLDR/P FILTPLOT COMLN STATUS GRPH.LB @FLIBB

C     Command Line:
C        FILTPLOT (/I or /M or /L) filename

C        where "filename" can be any legal RDOS filename

C        Either the I, M or L switch must be attached and indicates
C        an impulse, magnitude or log magnitude plot, respectively.

C*******************************************************************************

      REAL RFILT(1024),RP1,RP2,RP3
      INTEGER FILE(7),SP,MS(2),RET,NUMBLK,LASTBYT,POINTS,BYTS
      LOGICAL ITEST,SET
C
C     Retrieve command line file and verify only one.
C
      CALL COMLN(RET,SP,FILE,SP,SP,MS,SP,SP,SP)
      IF (RET.EQ.1) GO TO 20
      CALL ERROR("incorrect command line syntax")
C
C     Verify input file exists and retrieve it's contents.
C
 20   CALL STATUS(FILE,NUMBLK,LASTBYT)
      POINTS=(NUMBLK*128)+(LASTBYT/4)
      BYTS=POINTS*4
      CALL FOPEN(1,FILE)
      CALL RDSEQ(1,RFILT,BYTS,IER)
      IF (IER.NE.1) CALL ERROR("RDSEQ error")
      CALL FCLOSE(1)
C
C     Determine the type of plot.
C
```

```
      SET=ITEST(MS(1),7)        ;check for I switch
      IF (SET) GO TO 50

      SET=ITEST(MS(1),3)        ;check for M switch
      IF (SET) GO TO 30

      SET=ITEST(MS(1),4)        ;check for L switch
      IF (SET) GO TO 40
      CALL ERROR("invalid command line switch")
C
C     Plot the first half of the magnitude response.
C
 30   POINTS=POINTS/2
      GO TO 60
C
C     Compute log magnitude response.
C
 40   POINTS=POINTS/2
      DO 16 I=1,POINTS
      RFILT(I)=10.*ALOG10(RFILT(I))
 16   CONTINUE
      GO TO 60
C
C     Plot impulse response with vertical lines.
C
 50   IF (POINTS.GT.512) CALL ERROR("impulse response too long")
      CALL GRPH2(FILE,1,RFILT,RP1,POINTS,1,RP2,RP3,0)
      ACCEPT                    ;allow user to position cursor for typing
      ACCEPT                    ;on graph
      ACCEPT
      ACCEPT
      GO TO 90
C
C     Plot magnitude and log magnitude response with smooth line.
C
 60   IF (POINTS.NE.512) CALL ERROR("frequency response not 1024 points")
      CALL GRPH2(FILE,1,RFILT,RP1,POINTS,0,RP2,RP3,0)
      ACCEPT
      ACCEPT
      ACCEPT
      ACCEPT

 90   CALL EXIT
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: Header
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine prints on the printer a header specifying an Eclipse
C        A/D/A conversion operation. The conversion results specified can
C        then be printed beneath the header.

C     Compile command:
C        FORTRAN HEADER

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        DEVICE          21 for A/D or 23 fo D/A

C        SPEC1           starting channel for A/D or D/A

C        SPEC2           ending channel for A/D or mode set for D/A

C        IDATA2          conversion count

C        IER             DOITW error return

C        IORBA           the operation's IORBA array

C        CLOCK           conversion count

C*******************************************************************************

      SUBROUTINE HEADER(DEVICE,SPEC1,SPEC2,IDATA2,IER,IORBA,CLOCK)

      INTEGER DEVICE,SPEC1,SPEC2,IDATA2,IER,IORBA(16),CLOCK

      IF (DEVICE.EQ.21 .OR. DEVICE.EQ.23) GO TO 605
      CALL ERROR("improper device number")

 605  CALL FGDAY (IMON, IDAY, IYR)
      CALL FGTIME (IHOUR, IMIN, ISEC)

      WRITE (12,10)
 10   FORMAT (1X,"Eclipse A/D/A operation")
      WRITE (12,115)
      WRITE (12,11) IMON,IDAY,IYR
 11   FORMAT (1X,"date: ",I2,"/",I2,"/",I2)
      WRITE (12,12) IHOUR,IMIN
 12   FORMAT (1X,"time: ",I2," : ",I2)
      WRITE (12,115)
      WRITE (12,1)
      IF (CLOCK.EQ.1) WRITE (12,21)
```

```
      IF (CLOCK.EQ.2) WRITE (12,24)
      IF (CLOCK.EQ.3) WRITE (12,23)
      IF (CLOCK.EQ.4) WRITE (12,22)
      WRITE (12,3) SPEC1
      IF (DEVICE.EQ.21) WRITE(12,4) SPEC2
      IF (DEVICE.EQ.23) WRITE(12,8) SPEC2
      WRITE (12,5) IDATA2
      WRITE (12,6) IER
      WRITE (12,7)
      WRITE (12,9) (IORBA(I),I=1,16)
1     FORMAT (1X,"analog-to-digital conversion")
20    FORMAT (1X,"digital-to-analog conversion")
2     FORMAT (1X,"Clock: ",I2)
3     FORMAT (1X,"First channel: ",I2)
4     FORMAT (1X,"Last channel: ",I2)
5     FORMAT (1X,"Conversion count: ",I5)
8     FORMAT (1X,"Mode: ",I2)
6     FORMAT (1X,"DOIT error: ",I4)
7     FORMAT (1X,"Iorba(1-16) (Octal format):")
9     FORMAT (1X,16(1X,O6))
21    FORMAT (1X,"pulse clock")
22    FORMAT (1X,"DCH clock")
23    FORMAT (1X,"internal clock")
24    FORMAT (1X,"external clock")
      WRITE (12,115)
115   FORMAT (1X)

      RETURN
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: InFile
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine reads a specified section of a disk file into a real
C        data array.

C     Compile command:
C        FORTRAN INFILE

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        FILENAM                 the disk filename (in S format) to be read

C        STBLK                   the number of the disk block to begin
C                                reading (the first block of a file is 0)

C        NUMBLK                  the number of disk blocks to read

C        ARRAY                   the array to receive data

C        LEN                     the length of the data array

C*******************************************************************************

      SUBROUTINE INFILE(FILENAM,STBLK,NUMBLK,ARRAY,LEN)

      INTEGER FILENAM(7),STBLK,NUMBLK,LEN
      REAL ARRAY(LEN)

      CALL OPEN(1,FILENAM,1,IER)
      IF (IER.NE.1) WRITE(10,1) IER,FILENAM(1)
 1    FORMAT("
     *OPEN error ",I6," with file",S13)

      CALL RDBLK(1,STBLK,ARRAY,NUMBLK,IER)
      IF (IER.NE.1) WRITE(10,2) IER,FILENAM(1)
 2    FORMAT("
     *RDBLK error ",I6," with file ",S13)

      CALL FCLOSE(1)
      RETURN
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: LenChc
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine verifies that a disk file fits a specified minimum
C        or maximum size. If the disk file is too large or small, the
C        program is halted and an error message is printed on the console's
C        screen. The unit used to measure the file's length is a real
C        number element, which requires 4 bytes of memory.

C     Compile command:
C        FORTRAN LENCHC

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        FILENAM                 the disk filename (in S format) that is
C                                being checked

C        NUMBLK                  the number of the last disk block of the
C                                disk file

C        LASTBYT                 the  number of the last byte in the last
C                                disk block of the file

C        MIN                     the minimum acceptable number of real elements

C        MAX                     the maximum acceptable number of real elements

C*******************************************************************************

      SUBROUTINE LENCHC(FILENAM,NUMBLK,LASTBYT,MIN,MAX)

      INTEGER FILENAM(7),NUMBLK,LASTBYT,LEN,MAX,MIN

      LEN=(NUMBLK*128)+(LASTBYT/4)
      IF (LEN.LT.MIN) WRITE(10,1) FILENAM(1),MIN
      IF (LEN.GT.MAX) WRITE(10,2) FILENAM(1),MAX
      IF (LEN.LT.MIN .OR. LEN.GT.MAX) GO TO 20
 1    FORMAT("
     *File ",S13," must contain at least ",I6," real elements.")
 2    FORMAT("
     *File ",S13," cannot contain over ",I6," real elements.")
      RETURN

 20   TYPE "
     *program aborted"
      CALL EXIT
      END
```

C**********

```
C*******************************************************************************

C     Title: NewScr
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine erases the screen by typing 24 blank lines.

C     Compile command:
C        FORTRAN NEWSCR

C*******************************************************************************

      SUBROUTINE NEWSCR

      DO 10 I=1,24
      TYPE
 10   CONTINUE

      RETURN
      END

C*******************************************************************************
```

```
C*************************************************************************

C     Title: Paper
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine prints sections of an integer data array on the
C        printer in 512-word pages. The calling program specifies all
C        of the parameters required.

C        This routine was designed for printing data collected with the
C        Eclipse A/D/A device. When executing the real number print
C        option, the integer word is converted to the real number
C        equivalent that this device uses to store data samples.

C     Compile command:
C        FORTRAN PAPER

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        IFOR           display format: 1 for integer, 2 for real number
C                       and 3 for octal

C        ISTART         the starting page

C        ISTOP          the ending page

C        ARRAY          the data array to be shown

C        LEN            the length of the data array

C*************************************************************************

      SUBROUTINE PAPER(IFOR,ISTART,ISTOP,ARRAY,LEN)

      INTEGER IFOR,ISTART,ISTOP,LEN,ARRAY(LEN),IPRT,IPAGE
      REAL TOPVOLT,REALNUM

      TOPVOLT=5.0   ;magnitude of Eclipse device bi-polar setting
      IPRT=32

      IPAGE=ISTART-1
      I1=(ISTART-1)*512
  610 I2=0
      IPAGE=IPAGE+1
      WRITE (12,8) IPAGE,IPRT
      WRITE (12,115)
  115 FORMAT (1X)
  8   FORMAT (1X,"page",I3,"   of",I3)
  615 I3=0
  620 I4=0
```

```
625   I1=I1+1
      I4=I4+1
      REALNUM=FLOAT(ARRAY(I1))/32768.0*TOPVOLT    ;convert to real number
      IF (IFOR.EQ.1) WRITE (12,9) ARRAY(I1)
      IF (IFOR.EQ.2) WRITE (12,14) REALNUM
      IF (IFOR.EQ.3) WRITE (12,13) ARRAY(I1)
14    FORMAT ("+",1X,F7.4,Z)
13    FORMAT ("+",1X,I6,Z)
9     FORMAT ("+",1X,O6,Z)
      IF (I4.NE.16) GO TO 625
      WRITE (12,115)
      I3=I3+1
      IF (I3.NE.16) GO TO 620
      WRITE (12,115)
      WRITE (12,115)
      I2=I2+1
      IF (I2.NE.2) GO TO 615
      IF (IPAGE.NE.ISTOP) GO TO 610

      RETURN
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: Plot
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This program allows the user to set the plotting options in the
C        GRPH2 subroutine to plot real and complex data files.

C     Compile command:
C        FORTRAN PLOT

C     Load command:
C        RLDR/P PLOT INFILE STATUS GRPH.LB @FLIB@

C     Enviroment:
C        This is a Fortran V program that has been designed to run from a
C        Tektronix graphics terminal on a mapped-RDOS Eclipse S/250
C        minicomputer system.

C*******************************************************************************

      REAL RDATA(512),IDATA(512),TEMP(1024),SP1,SP2
      INTEGER FILENAM(7),FIRST,NUMB,BLKS,BYTS,ITYPE,LEN,TOTBLKS,POINTS
      INTEGER IPLO,IDEC,ISC,IOP,IAN,IMO

      CALL ERS(1)
      CALL FDELAY(10)

      TYPE "(CR)
     *This program plots up to 512 specified points(CR)
     *from file on the tetronix graphics terminal."
      ISEC=2
      ISC=0

 30   ACCEPT "(CR)
     *Enter filename for reading:"
      READ(11,5) FILENAM(1)
 5    FORMAT(S13)

      IF (ISEC.EQ.1) GO TO 50

 40   ACCEPT "(CR)
     *What type of data does this file contain,(CR)
     *   1: real(CR)
     *   2: complex(CR)
     *selection:",ITYP

      IF (ITYP.EQ.1 .OR. ITYP.EQ.2) GO TO 50
      WRITE(10,1)
 1    FORMAT ("(CR)(CR)(CR)
     *Please chose only from the options given.")
      GO TO 40
```

```
50    CALL STATUS(FILENAM,BLKS,BYTS)
      LEN=128/ITYP

      TOTBLKS=4*ITYP
      IF (BYTS.EQ.512) BLKS=BLKS+1

60    WRITE(10,2) FILENAM(1),BLKS,BLKS
2     FORMAT("(CR)
     *File ",S13," contains ",I3," diskblocks, numbered from 1 - ",I3,".")
      WRITE(10,3) LEN
3     FORMAT("(CR)
     *Each disk block contains ",I3," elements.")
      WRITE(10,4) TOTBLKS
4     FORMAT("(CR)
     *Up to ",I1," disk blocks can be plotted in.")

      ACCEPT "(CR)
     *Please specify,(CR)
     *   starting block:",FIRST
      IF (ISEC.EQ.2) ACCEPT "
     *   number of blocks:",NUMB
      CALL ERS(1)
      CALL FDELAY(10)

      IF ((FIRST-1).GT.BLKS) GO TO 60
      IF (NUMB.GT.TOTBLKS) GO TO 60
      IF ((NUMB+FIRST-2).GT.BLKS) GO TO 60

      FIRST=FIRST-1

      IF (ITYP.EQ.1) GO TO 70

C*************************************************************************

      CALL INFILE(FILENAM,FIRST,NUMB,TEMP,1024)

      K=0
      DO 72 I=1,512
      K=K+1
      RDATA(I)=TEMP(K)
      K=K+1
      IDATA(I)=TEMP(K)
72    CONTINUE

74    ACCEPT "(CR)
     *Which data plot(s) will be viewed,(CR)
     *   1: real data(CR)
     *   2: imaginary data(CR)
     *   3: both(CR)
     *selection:",IPLO
      IF (IPLO.GE.1 .AND. IPLO.LE.3) GO TO 80
      WRITE(10,1)
      GO TO 74
```

```
 70   IF (ISEC.EQ.2) CALL INFILE(FILENAM,FIRST,NUMB,RDATA,512)
      IF (ISEC.EQ.1) CALL INFILE(FILENAM,FIRST,NUMB,IDATA,512)
      IPLO=1
      IF (ISEC.EQ.1) IPLO=3
      IF (ISEC.EQ.1) GO TO 80

C*************************************************************************

 75   ACCEPT "(CR)
     *Do you want to place a second plot(CR)
     *of real data on the graph?(CR)
     *   1: yes(CR)
     *   2: no(CR)
     *selection:",ISEC

      IF (ISEC.EQ.1) GO TO 30
      IF (ISEC.EQ.2) GO TO 80
      WRITE(10,1)
      GO TO 75

C*************************************************************************

 80   ACCEPT "(CR)
     *Do you want to set the scaling limits?(CR)
     *   1: yes(CR)
     *   2: no(CR)
     *selection: ",IOP

      IF (IOP.EQ.1) GO TO 81
      IF (IOP.EQ.2) GO TO 83
      WRITE (10,1)
      GO TO 80

 81   ACCEPT "(CR)
     *Enter the maximum: ",SP2
      ACCEPT "
     *Enter the minimum: ",SP1
      ISC=1

C*************************************************************************

 83   ACCEPT "(CR)
     *Do you want to,(CR)
     *   1: connect with vertical lines(CR)
     *   2: connect with smooth line(CR)
     *selection: ",IAN

      IF (IAN.EQ.1) IMO=1
      IF (IAN.EQ.2) IMO=0
      IF (IAN.EQ.1 .OR. IAN.EQ.2) GO TO 85
      WRITE (10,1)
      GO TO 83

C*************************************************************************
```

```
 85   ACCEPT "(CR)
     *Enter the number of points to plot: ",POINTS
      TYPE "ISC",ISC,"SP1",SP1,"SP2",SP2
      ACCEPT
      IF (IPLO.EQ.1) CALL GRPH2(FILENAM,1,RDATA,IDATA,POINTS,IMO,SP1,SP2,ISC)
      IF (IPLO.EQ.2) CALL GRPH2(FILENAM,1,IDATA,RDATA,POINTS,IMO,SP1,SP2,ISC)
      IF (IPLO.EQ.3) CALL GRPH2(FILENAM,2,IDATA,RDATA,POINTS,IMO,SP1,SP2,ISC)
      ISC=0

      ACCEPT
      ACCEPT
      ACCEPT
      ACCEPT
      CALL ERS(1)
      CALL FDELAY(10)

 90   ACCEPT "(CR)
     *Do you want to,(CR)
     *   1: plot from another file(CR)
     *   2: plot from current file(CR)
     *   3: exit(CR)
     *selection:",IDEC
      ISEC=2
      IF (IDEC.EQ.1) GO TO 30
      IF (IDEC.EQ.2) GO TO 60
      IF (IDEC.EQ.3) GO TO 100
      WRITE(10,1)
      GO TO 90

 100  CALL EXIT
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: RdByts
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine reads a section of data from disk file into an
C        integer data array.

C     Compile command:
C        FORTRAN RDBYTS

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        FILE                    the disk filename (in S format) to be read

C        BYTS                    the number of byts to be read

C        ARRAY                   the array to receive data

C        LEN                     the length of the data array

C*******************************************************************************

      SUBROUTINE RDBYTS(FILE,BYTS,ARRAY,LEN)

      INTEGER FILE(7),BYTS,LEN
      INTEGER ARRAY(LEN)

      CALL OPEN(1,FILE,1,IER)
      IF (IER.NE.1) WRITE(10,1) IER,FILE(1)
 1    FORMAT("
     *OPEN error ",I6," with file ",S13)

      CALL RDSEQ(1,ARRAY,BYTS,IER)
      IF (IER.NE.1) WRITE(10,2) IER,FILE(1)
 2    FORMAT("
     *RDSEQ error ",I6," with file ",S13)

      CALL FCLOSE(1)
      RETURN
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: RedBuf
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine reads a section of disk file into an integer data
C        array. The file and data section are specified interactively
C        by the user.

C     Compile command:
C        FORTRAN REDBUF

C     Comments:
C        The variables ARRAY and LEN that are passed to this routine are
C        the data array and it's length, respectively. On return, the array
C        contains the user data.

C*******************************************************************************

      SUBROUTINE REDBUF(ARRAY,LEN)

      INTEGER LEN,ARRAY(LEN),FILENAM(7),IFIRST,INUM,IDEC

 500  TYPE
      ACCEPT "
     *Enter the filename for reading:"
      READ (11,2) FILENAM(1)
 2    FORMAT (S13)

      CALL OPEN (1,FILENAM,2,IER)
      IF (IER.EQ.13) GO TO 510
      IF (IER.NE.1) TYPE "OPEN error",IER

      ACCEPT "(CR)
     *Enter the starting block for reading,(CR)
     *   (the first block of a file is 1):",IFIRST
      IFIRST=IFIRST-1

      ACCEPT "(CR)
     *Enter the number of blocks for reading:",INUM

      CALL RDBLK(1,IFIRST,ARRAY,INUM,IER)
      IF (IER.NE.1) TYPE "RDBLK error",IER
      IF (IER.NE.1) GO TO 520
      CALL RESET
      GO TO 100

 510  TYPE "(CR)
     *This file does not exist."
      GO TO 520

 520  CALL RESET
```

```
      ACCEPT "(CR)
     #Do you want to,(CR)
     #   1: try another file(CR)
     #   2: return to the main menu(CR)
     #selection:",IDEC

      IF (IDEC.EQ.1) GO TO 500
      IF (IDEC.EQ.2) GO TO 100
      WRITE (10,1)
1     FORMAT("(CR)(CR)(CR)
     #Please make selections only from the given options.")
      GO TO 520

100   RETURN
      END

C*******************************************************************************
```

```
C*************************************************************************

C     Title: SeeIt
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine displays sections of an integer data array on the
C        screen in 128-word pages. The calling program specifies all the
C        parameters required.

C        This routine was designed for displaying data collected with the
C        Eclipse A/D/A device. When executing the real number display
C        option, the integer word is converted to the real number
C        equivalent that this device uses to store data samples.

C     Compile command:
C        FORTRAN SEEIT

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        IFOR           display format: 1 for integer, 2 for real number
C                       and 3 for octal

C        ISTART         the starting page

C        ISTOP          the ending page

C        ARRAY          the data array to be shown

C        LEN            the length of the data array

C*************************************************************************

      SUBROUTINE SEEIT(IFOR,ISTART,ISTOP,ARRAY,LEN)

      INTEGER IFOR,ISTART,ISTOP,LEN,ARRAY(LEN),ITOT,IPAGE
      REAL REALNUM,TOPVOLT

      ITOT=128
      TOPVOLT=5.   ;magnitude of Eclipse device bi-polar setting

  505 TYPE "(CR)(CR)
     *Press carriage return to begin and(CR)
     *to continue with the next page.(CR)"
      ACCEPT

      IPAGE=ISTART-1
      I1=(ISTART-1)*128
  510 I2=0
      IPAGE=IPAGE+1
      TYPE "(CR) page",IPAGE,"         of",ITOT,"(CR)"
```

```
515   I3=0
520   I4=0
525   I1=I1+1
      I4=I4+1
      REALNUM=FLOAT(ARRAY(I1))/32768.0*TOPVOLT   ;convert to real number
      IF (IFOR.EQ.1) WRITE (10,110) ARRAY(I1)
      IF (IFOR.EQ.2) WRITE (10,111) REALNUM
      IF (IFOR.EQ.3) WRITE (10,112) ARRAY(I1)
110   FORMAT (1X,O6,Z)
111   FORMAT (1X,F7.4,Z)
112   FORMAT (1X,I6,Z)
      IF (I4.NE.8) GO TO 525
      WRITE (10,115)
115   FORMAT (1X)
      I3=I3+1
      IF (I3.NE.8) GO TO 520
      WRITE (10,115)
      WRITE (10,115)
      I2=I2+1
      IF (I2.NE.2) GO TO 515
      ACCEPT
      IF (IPAGE.NE.ISTOP) GO TO 510

      RETURN
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: SetUp
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This is a special purpose routine used by program INDIGI and
C        OUTDIGI. It allows the user to select the type of format and
C        section of data buffer for printing/displaying.

C     Compile command:
C        FORTRAN SETUP

C     Comments:
C        The variable IOP that is passed to this routine has the value 2,
C        for data buffer display, or 3, for data buffer print.

C        The other variable values are returned to the calling program
C        as set by the user.

C*******************************************************************************

      SUBROUTINE SETUP(IFOR,IOP,ISTART,ISTOP)

 230  ACCEPT "(CR)
     #What type of format?(CR)
     #   1: two's complement(CR)
     #   2: real number(CR)
     #   3: integer number(CR)
     #selection:",IFOR

      IF (IFOR.LT.1) GO TO 230
      IF (IFOR.GT.3) GO TO 230
 231  IF (IOP.EQ.2) GO TO 225
      IF (IOP.EQ.3) GO TO 235

 225  TYPE "(CR)
     #There are 128 pages of data, numbered 1 through 128,(CR)
     #with each page containing 128 samples."
      GO TO 250

 235  TYPE "(CR)
     #There are 32 pages of data, numbered 1 through 32,(CR)
     #with each page containing 512 samples."
 250  ACCEPT "(CR)
     #What page will be first? ",ISTART
      ACCEPT "
     #What page will be last? ",ISTOP

      IF (ISTART.LT.1) GO TO 231
      ITEST=((-96*IOP)+320)
      IF (ISTOP.GT.ITEST) GO TO 231
      IF (ISTART.GT.ISTOP) GO TO 231
```

```
      RETURN
      END

C*************************************************************************
```

```
C*******************************************************************************

C     Title: Sort2
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine receives two filenames and their corresponding switch
C        arrays. It arranges the filenames and switch arrays in order of
C        position according to specified switch values also passed to it.

C     Compile command:
C        FORTRAN SORT2

C     Command line:
C        CALL SORT2(X,Y,FILE1,FILE2,F1,F2)

C        where,

C        X and Y are numbers corresponding to switch options set in
C        F1 and F2. The number is the position of the letter in the
C        alphabet, that is A=1...Z=26.

C        FILE1/F1 and FILE2/F2 are filenames and their corresponding
C        switch arrays in the format returned by the COMARG call (this
C        is S format for the filenames).

C        On return, the file with switch X set will occupy the position
C        of FILE1/F1 and the file with switch Y set will occupy the
C        position of FILE2/F2.

C     Comments:
C        F1 and F2 may also contain other switch values besides X and Y,
C        and these values will not be altered.

C        If one or both of the files do not contain either of the switch
C        values specified, the program is halted with an error message.

C        If either file contains both switch values X and Y, the program
C        is halted with an error message.

C*******************************************************************************

      SUBROUTINE SORT2(X,Y,FILE1,FILE2,F1,F2)

      INTEGER X,Y,FILE1(7),FILE2(7),F1(2),F2(2),TEMP(7),BIT

      LOGICAL ITEST,CASE1,CASE2
C
C     The first element of the switch array contains switches A-P.
C     The second element of the switch array contains switches Q-Z.
C
      I=0
      IF (X.GT.1 .AND. X.LE.16) I=1    ;determine which switch array
```

```
      IF (X.GT.16 .AND. X.LE.26) I=2  ;element would contain switch X
      IF (I.EQ.0) GO TO 90
C
C     The following transformation gives the bit position of a
C     switch in the switch array element.
C
      BIT=(-1*X)+(16*I)
      CASE1=ITEST(F1(I),BIT)   ;test both switch arrays
      CASE2=ITEST(F2(I),BIT)   ;for switch X

      IF (CASE1.AND..NOT.CASE2) GO TO 60   ;switch X is in the first file
                                           ;and not second file
      IF (CASE2.AND..NOT.CASE1) GO TO 50   ;switch X is in the second file
                                           ;and not first file, so the
                                           ;file positions are switched
      GO TO 90

 50   DO 15 I=1,7
      TEMP(I)=FILE2(I)
      FILE2(I)=FILE1(I)
      FILE1(I)=TEMP(I)
 15   CONTINUE
      DO 16 I=1,2
      TEMP(I)=F2(I)
      F2(I)=F1(I)
      F1(I)=TEMP(I)
 16   CONTINUE

 60   I=0
      IF (Y.GT.1 .AND. Y.LE.16) I=1   ;determine which switch array
      IF (Y.GT.16 .AND. Y.LE.26) I=2  ;element would contain switch Y
      IF (I.EQ.0) GO TO 90

      BIT=(-1*Y)+(16*I)
      CASE1=ITEST(F1(I),BIT)   ;test both switch arrays
      CASE2=ITEST(F2(I),BIT)   ;for switch Y

      IF (CASE2.AND..NOT.CASE1) GO TO 100   ;switch Y is in second file
                                            ;and not first file

 90   TYPE "(CR)
     *The files included in the command line do not have(CR)
     *valid switches. Consult program documentation for(CR)
     *the correct syntax."
      STOP

 100  RETURN
      END

C*************************************************************************
```

```
C*************************************************************************
C     Title: Sort3
C     Author: Lt Allen
C     Date: Dec 82

C     Function: This subroutine arranges three files and their switches
C        in a specified order of position.

C     Compile command:
C        FORTRAN SORT3

C     Command line:
C        CALL SORT3(X,Y,Z,FILE1,FILE2,FILE3,F1,F2,F3)

C        where,

C        X,Y and Z are numbers corresponding to switch options set in
C        switch arrays F1,F2 and F3. The number is the position of
C        the letter in the alphabet, that is A=1...Z=26.

C        FILE1/F1,FILE2/F2 and FILE3/F3 are filename arrays and
C        their corresponding switch arrays in the format returned by
C        the COMARG call (this is S format for the filenames).

C        On return, the file with switch X set will occupy the position
C        of FILE1/F1, the file with switch Y set will occupy the
C        position of FILE2/F2 and the file with switch Z set will
C        occupy the position of FILE3/F3.

C     Comments:
C        F1,F2 and F3 may also contain other switch values besides X,Y
C        and Z, and these values will not be altered.

C        If one or more of the files do not contain any of the switch
C        values or if any of the files contains more than one of the
C        switch values, the program is halted with an error message.

C*************************************************************************

      SUBROUTINE SORT3(X,Y,Z,FILE1,FILE2,FILE3,F1,F2,F3)

      INTEGER X,Y,Z,FILE1(7),FILE2(7),FILE3(7),F1(2),F2(2),F3(2)
      INTEGER TEMP(7),BIT

      LOGICAL ITEST,CASE1,CASE2,CASE3
C
C     The first element of the switch array contains switches A-P.
C     The second element of the switch array contains switches Q-Z.
C
      I=0
      IF (X.GE.1 .AND. X.LE.16) I=1   ;determine which switch array
      IF (X.GT.16 .AND. X.LE.26) I=2  ;element would contain switch X
      IF (I.EQ.0) GO TO 95
C
```

```
C     The following transformation gives the bit position of a switch
C     array element.
C
      BIT=(-1*X)+(16*I)
      CASE1=ITEST(F1(I),BIT)
      CASE2=ITEST(F2(I),BIT)
      CASE3=ITEST(F3(I),BIT)
C
C     First, check if switch X is in the first file and not the other two.
C
      IF ((CASE1.AND..NOT.CASE2) .AND. .NOT.CASE3) GO TO 70
C
C     Second, check if switch X is in the second file and not the other two.
C
      IF ((CASE2.AND..NOT.CASE1) .AND. .NOT.CASE3) GO TO 50
C
C     Third, check if switch X is in the third file and not the other two.
C
      IF ((CASE3.AND..NOT.CASE1) .AND. .NOT.CASE2) GO TO 60
      GO TO 95
C
C     Place the file with switch X in the first position.
C
 50   DO 15 I=1,7
      TEMP(I)=FILE2(I)
      FILE2(I)=FILE1(I)
      FILE1(I)=TEMP(I)
 15   CONTINUE
      DO 16 I=1,2
      TEMP(I)=F2(I)
      F2(I)=F1(I)
      F1(I)=TEMP(I)
 16   CONTINUE
      GO TO 70

 60   DO 17 I=1,7
      TEMP(I)=FILE3(I)
      FILE3(I)=FILE1(I)
      FILE1(I)=TEMP(I)
 17   CONTINUE
      DO 18 I=1,2
      TEMP(I)=F3(I)
      F3(I)=F1(I)
      F1(I)=TEMP(I)
 18   CONTINUE

 70   I=0
      IF (Y.GE.1 .AND. Y.LE.16) I=1   ;determine which switch array
      IF (Y.GT.16 .AND. Y.LE.26) I=2  ;element would contain switch Y
      IF (I.EQ.0) GO TO 95
C
C     Find switch Y's bit position.
C
      BIT=(-1*Y)+(16*I)
```

```
      CASE1=ITEST(F1(I),BIT)
      CASE2=ITEST(F2(I),BIT)
      CASE3=ITEST(F3(I),BIT)
C
C     First, check if switch Y is in the second file and not the other two.
C
      IF ((CASE2.AND..NOT.CASE1) .AND. .NOT.CASE3) GO TO 90
C
C     Second, check if switch Y is in the third file and not the other two.
C
      IF ((CASE3.AND..NOT.CASE2) .AND. .NOT.CASE1) GO TO 80
      GO TO 95
C
C     Place the file with switch Y in the second position.
C
 80   DO 19 I=1,7
      TEMP(I)=FILE3(I)
      FILE3(I)=FILE2(I)
      FILE2(I)=TEMP(I)
 19   CONTINUE
      DO 20 I=1,2
      TEMP(I)=F3(I)
      F3(I)=F2(I)
      F2(I)=TEMP(I)
 20   CONTINUE

 90   I=0
      IF (Z.GE.1 .AND. Z.LE.16) I=1   ;determine which switch array
      IF (Z.GT.16 .AND. Z.LE.26) I=2  ;element would contain switch Z
      IF (I.EQ.0) GO TO 95
C
C     Find switch Z's bit position.
C
      BIT=(-1*Z)+(16*I)
      CASE1=ITEST(F1(I),BIT)
      CASE2=ITEST(F2(I),BIT)
      CASE3=ITEST(F3(I),BIT)
C
C     Finally, check if switch Z is in the third file and not the other two.
C
      IF ((CASE3.AND..NOT.CASE1) .AND. .NOT.CASE2) GO TO 100

 95   TYPE "(CR)
     *The files included in the command line do not have(CR)
     *valid switches. Consult program documentation for(CR)
     *the correct syntax."
      STOP

 100  RETURN
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: Status
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine returns the number of the last disk block and the
C        number of bytes in the last disk block of a specified disk file.

C     Compile command:
C        FORTRAN STATUS

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        FILENAM                 the disk filename (in S format) to be
C                                checked

C        NUMBLK                  returns the number of the last disk block
C                                in the file, which is the number of disk
C                                blocks minus one

C        LASTBYT                 returns the number of bytes that are in
C                                the last block of the file

C*******************************************************************************

      SUBROUTINE STATUS(FILENAM,NUMBLK,LASTBYT)

      INTEGER FILENAM(7),ISTAT(18),NUMBLK,LASTBYT

      CALL STAT(FILENAM,ISTAT,IER)
      IF (IER.EQ.13) GO TO 20
      IF (IER.NE.1) WRITE(10,1) IER,FILENAM(1)
 1    FORMAT("
     *STAT error ",I2," with file ",S13)
      NUMBLK=ISTAT(9)
      LASTBYT=ISTAT(10)

      RETURN

 20   WRITE(10,2) FILENAM(1)
 2    FORMAT("
     *File ",S13," does not exist.(CR)
     *program aborted")
      CALL EXIT
      END

C*******************************************************************************
```

```
C*******************************************************************************

C     Title: ToFile
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine writes a real data array to disk file. It first
C        deletes/creates the file, so that the file will only contain
C        the data passed. The calling program should verify that it is
C        agreeable to delete any existing file before calling this
C        routine.

C     Compile command:
C        FORTRAN TOFILE

C     Comments:
C        The variables that are passed to this routine have the following
C        meaning,

C        FILENAM                 the disk filename (in S format) to be
C                                written to; it will be created as a
C                                random file

C        ARRAY                   the data array to be written to file

C        LEN                     the length of the data array

C*******************************************************************************

      SUBROUTINE TOFILE(FILENAM,ARRAY,LEN)

      INTEGER FILENAM(7),LEN,BLKS
      REAL ARRAY(LEN)

      BLKS=INT(LEN/128)

      CALL DFILW(FILENAM,IER)
      CALL CFILW(FILENAM,2,IER)
      IF (IER.NE.1) WRITE(10,1) IER,FILENAM(1)
 1    FORMAT("
     *CFILW error ",I2," with file ",S13)

      CALL OPEN(1,FILENAM,3,IER)
      IF (IER.NE.1) WRITE(10,2) IER,FILENAM(1)
 2    FORMAT("
     *OPEN error ",I2," with file ",S13)

      CALL WRBLK(1,0,ARRAY,BLKS,IER)
      IF (IER.NE.1) WRITE(10,3) IER,FILENAM(1)
 3    FORMAT("
     *WRBLK error ",I2," with file ",S13)

      CALL FCLOSE(1)
```

```
      RETURN
      END

C*****************************************************************************
```

```
C*************************************************************************

C     Title: Warnng
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This routine prints Eclipse A/D/A device warning messages to the
C        screen explaining to the user what to do for various error
C        conditions. It should be placed in an A/D/A program just before
C        the conversion operation is performed.

C     Compile command:
C        FORTRAN WARNNG

C     Comments:
C        The clock that has been chosen for the conversion operation is
C        sent to the routine in variable CLOCK.

C*************************************************************************

      SUBROUTINE WARNNG(CLOCK)

      INTEGER CLOCK

      TYPE "(CR)
     *The conversion operation can be safely aborted at this(CR)
     *time by typing CTRL-A."

      TYPE "(CR)
     *After the conversion operation has been initiated, wait an(CR)
     *appropriate amount of time before considering to abort an operation(CR)
     *which will not return. The only way to abort a conversion(CR)
     *operation that will not return is by typing CTRL-A. However,(CR)
     *this may result in crashing the system."

      IF (CLOCK.EQ.60000K) TYPE "(CR)
     *If the conversion operation does not return in an appropriate(CR)
     *amount of time, verify that the external clock is properly(CR)
     *connected. The clock can be reconnected once the conversion(CR)
     *operation begins."

      TYPE "(CR)
     *Press carriage return to begin the conversion operation."

      RETURN
      END

C*************************************************************************
```

```
C*******************************************************************************

C     Title: WrtBuf
C     Author: Lt Allen
C     Date: Dec 82

C     Function:
C        This is a special purpose routine used by program INDIGI and
C        OUTDIGI. It allows the user to write specified sections of the
C        data buffer to a disk file.

C     Compile command:
C        FORTRAN WRTBUF

C     Comments:
C        The variables ARRAY and LEN that are passed to this routine are
C        the data buffer and it's length, respectively.

C*******************************************************************************

      SUBROUTINE WRTBUF(ARRAY,LEN)

      INTEGER LEN,ARRAY(LEN),FILENAM(7)

 245  TYPE "(CR)
     *There are 64 disc blocks in the data buffer, numbered 1(CR)
     *through 64, with each block containing 256 samples."

      ACCEPT "(CR)
     *What block will be first? ",ISTART
      ACCEPT "
     *What block will be last? ",ISTOP

      IF (ISTART.LT.1) GO TO 245
      IF (ISTOP.GT.64) GO TO 245
      IF (ISTART.GT.ISTOP) GO TO 245
      ISTART=ISTART-1

 255  ACCEPT "
     *Enter the filename for writing:"
      READ (11,15) FILENAM(1)
 15   FORMAT (S13)

 260  CALL CFILW (FILENAM,2,IER)
      IF (IER.EQ.12) GO TO 265
      IF (IER.NE.1) TYPE "CFILW error ",IER," with your file"

      CALL OPEN (1,FILENAM,2,IER)
      IF (IER.NE.1) TYPE "OPEN error ",IER," with your file"
      CALL WRBLK(1,ISTART,ARRAY,ISTOP,IER)
      IF (IER.NE.1) TYPE "WRBLK error ",IER," with your file"
      CALL CLOSE (1,IER)
      IF (IER.NE.1) TYPE "CLOSE error ",IER," with your file"
      GO TO 280
```

```
265  ACCEPT "(CR)
     #This file already exists.(CR)(CR)
     #Do you want to,(CR)
     #   1: delete the current file(CR)
     #   2: try another file(CR)
     #selection:",IDEL

      IF (IDEL.EQ.1) GO TO 270
      IF (IDEL.EQ.2) GO TO 255
      WRITE (10,1)
1     FORMAT ("(CR)(CR)(CR)
     #Please make selections only from the given options.")
      GO TO 265

270   CALL DFILW (FILENAM,IER)
      IF (IER.NE.1) TYPE "DFILW error ",IER," with your file"
      GO TO 260

280   RETURN
      END

C*******************************************************************************
```

VITA

Gordon R. Allen was born on 16 January 1957 in Hardinsburg, Kentucky. He received the Bachelor of Science Electrical Engineering degree from the University of Kentucky in 1975. Upon graduation, he received a commission in the United States Air Force and was assigned to the Space and Missles Systems Organization, Los Angeles AFS, Los Angeles, California. In June 1981 he attended the Air Force Institute of Technology as a graduate student in the Digital Communications and Signal Processing Sequence. Gordon Allen is a member of Eta Kappa Nu and Tau Beta Pi.

Permanent address: Route 2, Box 229
Vine Grove, KY 40175

END

FILMED

3-83

DTIC